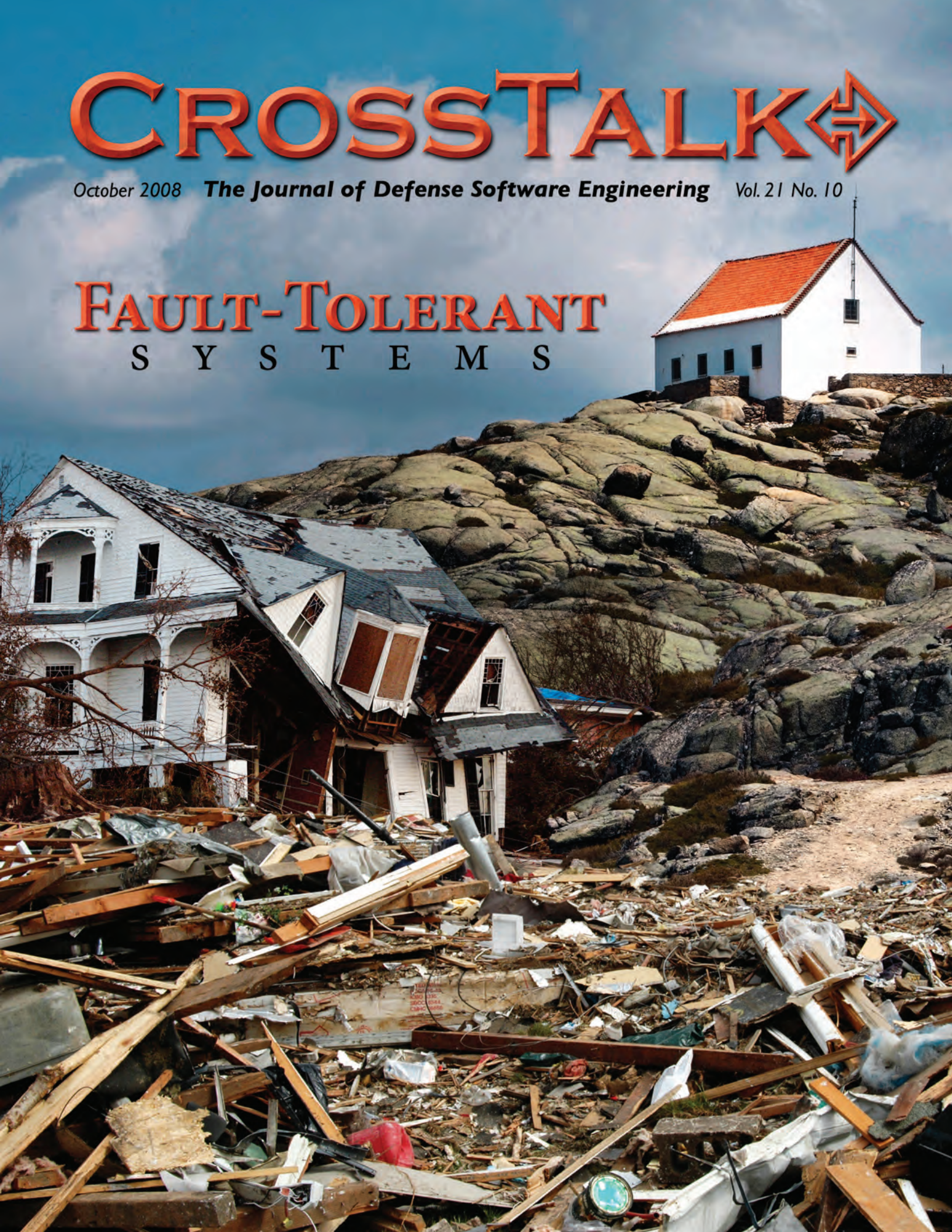
CrossTalk

October 2008 **The Journal of Defense Software Engineering** Vol. 21 No. 10

Fault-Tolerant Systems



Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
OCT 2008		**00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk. The Journal of Defense Software Engineering, Volume 21, Number 10	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS/MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**32**	

Development of Fault-Tolerant Systems

Software Engineering Technology

ON THE COVER

Cover Design by Kent Bingham

Additional art services provided by Janna Jensen

Departments

CROSSTALK

CO-SPONSORS:
DoD-CIO *The Honorable John Grimes*
OSD (AT&L) *Kristen Baldwin*
NAVAIR *Jeff Schwalb*
76 SMXG *Daniel Goddard*
309 SMXG *Karl Rogers*
DHS *Joe Jarzombek*

STAFF:
MANAGING DIRECTOR *Brent Baxter*
PUBLISHER *Kasey Thompson*
MANAGING EDITOR *Drew Brown*
ASSOCIATE EDITOR *Chelene Fortier-Lozancich*
PUBLISHING COORDINATOR *Nicole Kentta*
PHONE (801) 775-5555
E-MAIL stsc.customerservice@hill.af.mil
CROSSTALK ONLINE www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Department of Defense Chief Information Office (DoD-CIO); the Office of the Secretary of Defense (OSD) Acquisition, Technology and Logistics (AT&L); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). DoD-CIO co-sponsor: Assistant Secretary of Defense (Networks and Information Integration). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsors: Oklahoma City-Air Logistics Center (ALC) 76 Software Maintenance Group (SMXG); and Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal. CROSSTALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 24.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the author and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Development of Safety-Critical Software Systems

Since the development of the digital computer, software continues to play an important and evolutionary role in the operation and control of hazardous, safety-critical functions. The reluctance of the engineering community to relinquish human control of hazardous operations has diminished dramatically in the last 15 years. Today, digital computer systems have autonomous control over safety-critical functions in nearly every major technology, both commercially and within government systems. This revolution is due primarily to the ability of software to reliably perform critical control tasks at speeds unmatched by its human counterpart. Other factors influencing this transition include our ever-growing need and desire for increased versatility, greater performance capability, higher efficiency, and a decreased life-cycle cost.

In most instances, software can meet all of the above attributes of the system's performance when properly designed. The logic of the software allows for decisions to be implemented without emotion and with speed and accuracy. This has forced the human operator out of the control loop because they can no longer keep pace with the speed, cost effectiveness, and decision making process of the system.

According to the MIL-STD-882D, the main objective (or definition) of system safety engineering, which includes safety-critical software systems, is "the application of engineering and management principles, criteria, and techniques to optimize all aspects of safety within the constraints of operations effectiveness, time, and cost throughout all phases of the system life cycle."

The ultimate responsibility for the development of a "safe system" rests with program management. The commitment to provide qualified people and an adequate budget and schedule for a software development program begins with the program director or manager. Top management must be a strong voice of safety advocacy and must communicate this personal commitment to each level of program and technical management. Project directors or managers must support the integrated safety process between systems engineering, software engineering, and safety engineering in the design, development, testing, and operation of the system software.

This issue of CrossTalk provides an in-depth look at the implementation and development of safety-critical software systems. It also explores how these systems will likely face unplanned challenges during long-term development, requiring developers to build flexibility into their approaches.

Authors Dr. Victor Basili, Kathleen Dangle, Linda Esker, Frank Marotta, and Ioana Rus guide readers through their methodology for developing early safety measures on safety-critical software system projects in *Measures and Risk Indicators for Early Insight Into Software Safety*.

In *Safety and Security: Certification Issues and Technologies*, Dr. Benjamin M. Brosgol analyzes the two primary safety and security standards—DO-178B and the Common Criteria—and gives software professionals the tools to avoid hazards and vulnerabilities.

First responders who need a secure and mobile coordination and communication infrastructure during crisis will take special interest in Sugih Jamin's *WebBee: A Platform for Secure Mobile Coordination and Communication in Crisis Scenarios*.

In *Constructing Change-Tolerant Systems Using Capability-Based Design*, Dr. James D. Arthur and Ramya Ravichandar recognize the need for flexibility and provide readers with thought-provoking ideas on how a capability-based approach may be the answer to complex, large-scale systems that are hostile to change.

And, finally, don't miss *DoD Business Mission Area Service-Oriented Architecture to Support Business Transformation* by Dennis E. Wisnosky, Dimitry Feldshteyn, Wil Mancuso, Al (Edward) Gough, Eric J. Riutort, and Paul Strassman. They examine whether a service-oriented architecture (SOA) is the best fit for the Department of Defense's (DoD's) Business Mission Area (accounting for roughly half of the DoD Information Technology budget) and examine the DoD's SOA vision.

I hope you enjoy reading CrossTalk's variety of articles on how to better approach the development of safety-critical software systems. I certainly did.

Ken Chirkis
Naval Air Systems Command

Measures and Risk Indicators for Early Insight Into Software Safety

Dr. Victor Basili
University of Maryland and Fraunhofer Center – Maryland

Kathleen Dangle and Linda Esker
Fraunhofer Center – Maryland

Frank Marotta
U.S. Army Aberdeen Test Center

Ioana Rus
Honeywell Aerospace

Software contributes an ever-increasing level of functionality and control in today's systems. This increased use of software can dramatically increase the complexity and time needed to evaluate the safety of a system. Although the actual system safety cannot be verified during its development, measures can reveal early insights into potential safety problems and risks. An approach for developing early software safety measures is presented in this article. The approach and the example software measures presented are based on experience working with the safety engineering group on a large Department of Defense program.

The purpose of the system safety process is to identify and mitigate hazards associated with the operation and maintenance of a system under development. System safety is often implemented through an approach that identifies hazards and defines actions that will mitigate the hazard and verify that the mitigations have been implemented. The *residual risk* is the risk remaining when a hazard cannot be completely mitigated. The goal of the system safety process is to reduce this residual risk to an acceptable level, as defined by the safety certifier. Cost is a consideration in determining the level of acceptable residual risk.

As software contributes an ever-increasing level of functionality and control in today's systems, the system safety process must scrutinize software-specific components of the system. Software can contribute to system safety as both a hazard source and hazard mitigation. Software is not intrinsically hazardous but it plays a role in safety in many systems when it:

- Causes hardware to perform unsafe actions.
- Directs an operator to perform unsafe actions.
- Guides an operator to make unsafe decisions.
- Mitigates hazards.

In this article, we define a measurement approach that provides early visibility into the implementation of the software safety hazard process, assessing the level of consistency and discipline that is applied to the process for identifying and mitigating software-related hazards. Early process visibility assists safety engineers in detecting breakdowns in the process, asking the right kinds of questions, and making timely decisions that will improve the resulting system safety. This early visibility is important as mitigations typically affect system requirements and design; making these decisions late in the system development lifecycle can be cost-prohibitive. The proposed measurement approach identifies risks resulting from the application of the safety hazard analysis process (or lack thereof) by performing process checks and assesses the *potential* for achieving a safe system. It is important to note that this approach does not provide for an evaluation of the system's safety.

> *"Early process visibility assists safety engineers in detecting breakdowns in the process, asking the right kinds of questions, and making timely decisions that will improve the resulting system safety."*

This article begins by defining terms and documenting our assumptions. We then describe our approach for defining specific safety measures in the context of an existing environment and provide some examples.

Terminology and Key Concepts

A *hazard* is any real or potential condition that can cause injury, illness, or death to personnel; damage to or loss of a system, equipment, or property; or damage to the environment. Key terms associated with hazards and their management are:

- **Causes.** What can make the hazard occur?
- **Controls.** Mitigation actions whose purpose is to minimize the chances of a hazard occurring.
- **Verifications.** Some assurance, like safety test cases, that the hazard has been controlled.

A hazard is *open* if at least one of its causes is open; a cause is *open* if at least one of its controls is open; a control is *open* if at least one of its verifications is open. A hazard is *closed* when all of the controls for all of its causes have been implemented and verified.

A *safety-related requirement* is a requirement whose purpose is to control a hazard. One hazard might be addressed by several requirements (e.g., one hazard may affect several parts of the system), or one requirement might address several hazards (e.g., a central control or communication system may mitigate hazards from multiple nodes).

A *hazard tracking system* (HTS) is a repository of identified system hazards and their associated causes, controls, and verifications. Within the HTS, causes should be related with the system element causing the hazard, controls should be related with the requirement(s) controlling or mitigating the hazard, and verifications should be related with the hazard cause and the test verifying that the hazard is controlled.

A hazard is defined as a software-related hazard if it has at least one software cause or one software control. A software safety-related requirement is a software requirement that can create or contribute

to a hazard in the system or is defined to control or mitigate a hazard.

An example of a system hazard description that has a software-related cause is as follows:

- **Accident/Mishap.** Undesired and unplanned event that results in a specified level of loss (e.g., two planes collide).
- **Hazard/Description.** State that leads to an accident (e.g., guidance system may malfunction).
- **Hazard Cause.** The action causing the hazard to occur (e.g., a miscalculation of the projected trajectory).
- **Hazard Control or Safety Requirement.** Mitigation via a requirement or set of requirements whose purpose is to minimize the chances of a hazard (e.g., multiple computations of the projected trajectory are computed and polled).
- **Verification.** An assurance that the hazard has been controlled (e.g., safety test cases).

Figure 1 provides an illustration of the context for this example.

Several assumptions are made: (1) all hazards should be recorded in an HTS; (2) hazards are retired or have their associated risk reduced over time, but do not leave the HTS; and (3) closed hazards can become open hazards when a new cause is found. Although the approach does not prescribe a particular management or organizational structure, it is assumed that the safety and project organizations communicate and collaborate effectively in both evolving requirements and verifying mitigations. As the safety hazard analysis will impact requirements, design, code, and tests, it is assumed that the standard processes defined by the project for change management apply to artifacts impacted by safety hazard analysis.

The *level of rigor* (LoR) is the amount of requirements analysis, development discipline, testing, and configuration control required to mitigate the potential safety risks of the software component [1]. Each software component should be assessed and assigned an LoR for development. This refers to any mechanism put in place to treat specific requirements with special treatment, giving a piece of software higher levels of safety assurance and providing users higher confidence through greater discipline and process.

A *safety-related defect* is a defect that refers to a failure to comply with a safety requirement, an unexpected behavior that affects safety, or the recognition that a control has not been defined/implemented/verified. Safety-related defects should be traceable to one or more hazards or may generate new hazards. Defects can be counted directly or they can be weighed by the set of related requirements or hazards they affect. A *software defect tracking system* (i.e., tool/database to capture software defects identified during testing) is used as the source of safety-related software defects.

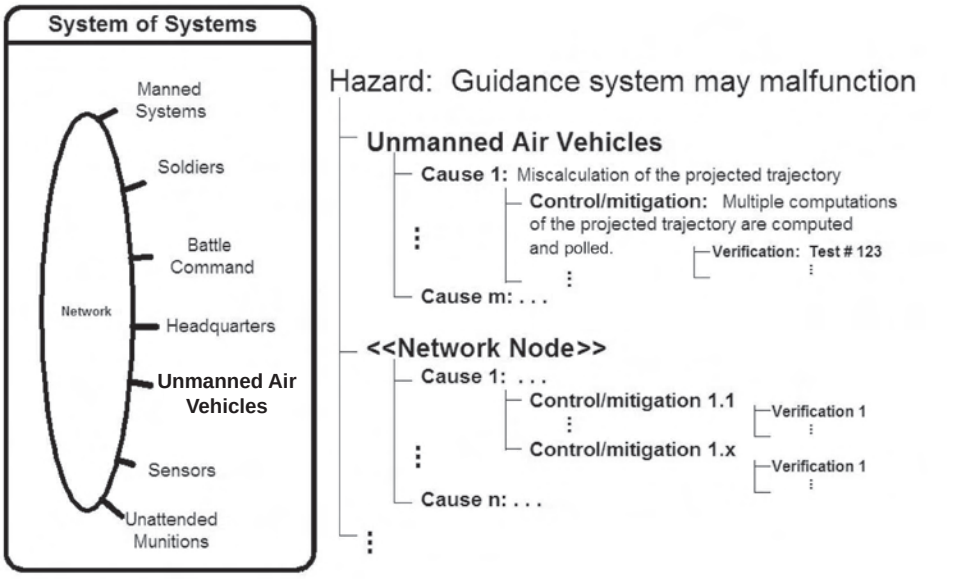


Figure 1: *Example of a Hazard, Cause, Control, and Verification*

Gaining Software Safety Visibility

Our goal in applying the proposed measurement approach is to provide software safety engineers visibility into the software safety process and to assist them in making judgments about the software safety process implementation and its execution. We identified five needs, and an associated inquiry area for each was defined:

1. **Software Safety Analysis Process.** Confirm that system and software requirements and development practices are in compliance with safety processes.
2. **Hazard and Mitigation Identification.** Ensure that the program is adequately identifying and documenting the appropriate information about a hazard (i.e., hazards, causes, and controls as defined by the software safety analysis process).
3. **Hazard Monitoring.** Ensure that sufficient actions are taken by analyzing and monitoring hazard causes, controls, and verifications over time (i.e., are the hazard controls being implemented and verified?).
4. **Appropriate LoR for Software Safety.** Balance risk with the cost of safety by identifying the appropriate software development LoR.
5. **Safety-Related Defects.** Identify whether any safety problems remain in the system for the safety assessment reports by identifying all outstanding safety-related defects.

For each area, readiness and visibility measures are defined, specifying different measurement details. A *readiness assessment* provides a preliminary view into the state of the safety process for software and checks that the data needed for the second type of measurement is available. *Software safety visibility* digs deeper by defining models, measures, and interpretations that provide information on the implementation of safety practices (or lack thereof) and points to safety-related risks and issues.

To minimize the overhead associated with data collection and analysis, a combination of a top-down goal/question/metric analysis and a bottom-up inventory of the data already collected by the organization is used to identify the measures that will be cost-effective and address management needs [2].

For example, to address software safety analysis, an investigation may be performed to determine whether there is a documented safety process that identifies requirements as safety-related and records that information in the requirements repository. If this is not true, then the program may have a problem and further measures that assume counting the number of safety-related requirements cannot be utilized. A sample set of key questions addressing the five inquiry areas for the readiness assessment are shown in Table 1 (see next page). All readiness questions must be answered *Yes* to indicate that the appropriate measure-

Inquiry Area	Readiness Assessment Questions
Software Safety Analysis Process	○ Is there a documented software safety process that identifies requirements as safety-related? ○ Are safety-related software requirements marked as such in the requirements repository?
Hazard and Mitigation Identification	○ Is there an (automated) HTS where software-related hazards, causes, controls, and verifications are recorded (and can be counted)?
Hazard Monitoring	○ Are hazards mapped back to their source (requirements) and controls mapped to requirements? ○ Are all the fields being entered into the HTS?
Appropriate LoR for Software Safety	○ Are the various levels of rigor identified and is the distribution rational?
Safety Defects	○ Are software safety-related failures/faults identified as such in the software defect tracking system? ○ Are safety-related test cases identified as such? ○ Are defect closures recorded?

Table 1: *Readiness Assessment Questions*

ments can be gathered. *No* answers provide an early warning that software safety may not be properly addressed. In this case, the recommended action is to identify why the data is not available (root cause) and take an appropriate corrective action. The questions in Table 1 address problems in dealing with safety in general and software safety in particular.

While these *data readiness* questions seem simplistic, they can uncover a host of issues that may not be obvious unless the questions are asked explicitly. These questions expose some common problems in implementing a useable, cost-effective HTS and in the overall hazard tracking approach:

- **Software Hazard Identification.** Safety-related requirements are not identified as such and hazard controls are not identified as software-related safety requirements if they are. This can demonstrate inadequate attention to software safety.
- **Hazard Traceability.** The HTS does not provide sufficient linkages among the requirements documentation system, the test plan, or to the defect tracking system. Hazards must be bi-directionally traceable to requirements, tests, and defects in order to verify complete coverage, determine comprehensiveness of the hazard analysis, and ensure that the hazard data represents the system accurately over time.
- **Data Integrity.** Hazards, causes, and controls may not be described in sufficient detail to be understood and verified. The information in the HTS must be accurate, clear, and specific in order to understand and track hazards throughout the development and deployment of the system.
- **LoR.** There may be difficulty in differentiating among different levels of rigor for the various software safety requirements and identifying, assigning, and tracking the appropriate LoR to specific software components that implement the safety-related requirement. Lack of proper LoR differentiation can lead to inadequate attention on high-risk hazards or too much attention on low-risk hazards. Additionally, the trade-off between higher levels of rigor and their associated higher costs must be considered in order to assess the *right* balance of LoR distribution. An LoR should be assigned and traceable from requirements through design to code.

Table 2: *Software Safety Visibility Needs*

Inquiry Area	Goal	Software Safety Visibility Questions
Software Safety Analysis Process	Check how well each organization, system, and integrator is addressing software safety in the system hazard analysis process.	○ Have a reasonable number of software safety-related requirements been identified?
Hazard and Mitigation Identification	Check if a reasonable number of software-related hazards, causes, controls, and verifications are identified.	○ Have a reasonable number of software safety hazards been identified? ○ Are causes, controls, and verifications being generated over time? ○ Does every cause have at least one control? ○ Does every control have at least one verification?
Hazard Monitoring	Check if software-related hazards (and hazard software components, i.e., causes, controls, and verifications) are identified and closed at an appropriate rate.	○ Have the number of open software causes/controls for hazards decreased over time?
Appropriate LoR for Software Safety	Check if the various software development groups are assigning reasonable levels of rigor to safety-related software.	○ Have the appropriate levels of rigor been allocated to software development?
Safety Defects	Check if software safety-related defects are being handled.	○ Have safety-related software defects been closed at a reasonable rate over time?

Many HTS problems are caused by an inadequate vision for the use of the HTS, such as when it is viewed as a storage repository rather than an analysis tool. It is important to make sure that (1) the HTS has adequate functionality, quality checks, and documentation; (2) there is traceability and synchronization among the various support systems (e.g., the HTS and the requirements management system and the defect tracking system); and (3) the quality of the data is monitored to minimize the need to scrub the data later on. The cost of not adhering to this advice is high rework costs and lower than desired system safety. Addressing these issues should simply be a part of the software safety development process.

Laying the Measurement Foundation

Once it is clear that the safety process has been established, deeper investigation of each inquiry area can be performed. An example set of software safety visibility

Inquiry Area	Measure(s)	Model(s)	Response(s)
Software Safety Analysis Process	Percent Software Safety Requirements (PSSR) Estimated PSSR (EPSSR) **PSSR =** # software safety requirements / # software requirements *100	***if* |PSSR - EPSSR| < e *then*** a reasonable number of software safety requirements have been identified ***where*** **the EPSSR =** the average of the PSSRs for all systems in the family, (*in line with other systems*) and **e** = σ (EPSSR) (i.e., standard deviation of the PSSRs used to calculate EPSSR) ***or*** **EPSSR** = #system safety requirements / #system requirements * 100, (*in line with system safety in general*) and **e** = 20% of EPSSR.	PSSR not being within the range of EPSSR should indicate the need for a management action. For example, check into the safety hazard elicitation process and whether it is being applied correctly, investigate the reason why the system under consideration has such a small (or large) percentage of safety requirements, and develop a "get well" plan. If the value is too large, what are the cost and schedule implications of corrective actions?
Hazard Monitoring	Hazard cause/control closure evolution (HCCE) $HCCE_{i,3} = MA_{i+1,3} / MA_{i,3}$ ***where*** $MA_{i,3} = (X_{i-2} + X_{i-1} + X_i)/ 3$ is the moving average of the set of open causes (controls) at three consecutive time intervals.	If $HCCE_{i,3} \geq 1$ then the closure rate of hazard software causes/controls is not converging.	If the number is ≥ 1 and it is not in the beginning phases of development, more effort should go into closing the hazard software causes/controls. If it is because the opens are increasing too fast (new hazards are being introduced, new causes for existing hazards), then investigate the reasons. If it is because the closes are not increasing fast enough, then investigate the reasons. Graphing the cumulative identified, open, and closed causes/controls provides good insight into the trends of these variables.
Safety Defects	Count by priority of open safety-related software trouble reports at time i (COSRTR).	If COSRTR ≠ 0 then there are open defects that need further analysis.	If all safety-related defects are not closed, then create a list of open defects, prioritize them, and investigate why they exist. This measure should be taken periodically starting at the beginning of test and up until safety assessment report delivery.

Table 3: *Some Examples of Software Safety Measures*

goals and questions is presented in Table 2. When a readiness assessment question has been satisfied, the software safety visibility questions and measures throughout the life cycle of the program can be applied.

Establishing the measures requires more than identifying the data to be collected. Each *measure* is characterized in terms of the *question* it answers, the *model* used to interpret its values in order to answer the target question, the *response* that suggests the action to be taken based upon the answer to the question, and the *scope* of applying the measure. Table 3 presents examples of *models* and *responses* for three of the five inquiry areas[1].

For each model, assumptions were made about how the resulting measurements should be interpreted. An *expected value* and a *range* are selected for within which the actual is acceptable. The expected value can be derived by: (1) historical data from past programs, (2) prior data from the current program, (3) proxy estimate (i.e., comparison with something similar), or (4) expert estimate. The range of the expected values can be based on general distributions or specific or related experience.

If the calculated value is not within the expected range, then there may be a problem. Expected values or ranges can be improved over time based upon the incorporation of new data into the model.

To illustrate these concepts, consider one measure proposed for the process area, PSSR, which is defined as *PSSR = # software safety requirements / # software requirements *100*. The model can be defined as:

***if* |PSSR – EPSSR| < e**
where EPSSR is the estimated value of PSSR, e is the acceptable threshold for deviation from the estimate, and (EPSSR -e, EPSSR +e) is the acceptable range, *then* a reasonable number of software safety requirements have been identified.

The key is to have good estimates for EPSSR and *e*. Ideally, historical data should be used and the estimated value and range (i.e., sigma, the standard deviation) is taken from a similar system or subsystem. However, there may be little historical data. In this case, proxies are identified for estimates[2].

One possible proxy is to use system safety requirements as the benchmark for software safety requirements. We can let the range be defined by some percentage around that value that provides initially acceptable limits. Once the program is under development, early data can be substituted on the program for these proxies. Thus:

EPSSR = #system safety requirements / #system requirements *100
and e = 20 percent of EPSSR.

The model is interpreted by defining a *response* if the resulting value is not within range. For example, if PSSR is not within the range *e* of EPSSR, it indicates the need for management action. One example would be to check into the safety analysis process and whether it is being appropriately applied, investigate the reason why the system under consideration has such a small (or large) percentage of safety requirements, and develop a *get well* plan.

In defining these measures, existing data sources (e.g., a hazard tracking database, requirements management repositories, and defect tracking systems) and processes (e.g., safety analysis processes) were leveraged. This can be done provided that the assumptions upon data collection (listed in the Terminology and Key Concepts section) are true. The derived measures in Table 3 can be graphically represented (e.g., as evolution over time), as appropriate, for the analysis results on the questions it helps to answer. Key issues for determining software safety visibility are: (1) selecting the right subset of measures, (2) defining appropriate thresholds, (3) determining appropriate management responses, and (4) providing user-friendly reports and actionable responses; all of these issues are program-dependent.

The safety measures collected by a program form the beginning of an experience base, which creates a historical base across current programs within a program and for future programs. To date, there is very little data on which to calibrate the models. It is hoped that programs will start collecting data so that more knowledge can be obtained and software safety measure baselines can be established.

Conclusion

The methodology presented here should be tailored to fit the context of the organization; it is not intended to imply a correct answer or an *all or nothing* approach. The areas of inquiry and the measures can be adjusted appropriately; however, as a minimum, any program dealing with safety should at least address the readiness questions.

Gaining visibility through objectives measures into software safety has become increasingly important for today's software-intensive programs. Although software safety measures cannot determine whether a system is safe, they can provide valuable indicators of problems and risks that give management critical knowledge for making timely and well-informed decisions.◆

References

1. Radio Technical Commission for Aeronautics, Inc. "Software Considerations in Airborne Systems and Equipment Certification." RTCA DO-178B. 1 Dec. 1992.
2. Basili, V., and D. Weiss. "A Methodology for Collecting Valid Software Engineering Data." IEEE Transactions on Software Engineering Nov. 1984: 728-738.

Notes

1. The *question* is omitted from this table due to space limitations; for *scope*, we assume that these measures apply to the entire system.
2. This argues for the need to accumulate data on programs, not just for the good of the current program but for use in future programs.

Additional Reading

1. Joint Software System Safety Committee. Software System Safety Handbook. Dec. 1999.
2. MIL-STD-882. DoD Standard Practice for System Safety. 10 Feb. 2000 <http://safetycenter.navy.mil/instructions/osh/milstd882d.pdf>.

About the Authors

Victor Basili, Ph.D., is a professor of computer science at the University of Maryland and Chief Scientist at the Fraunhofer Center – Maryland (FC-MD). He works on measuring, evaluating, and improving software processes and products.

E-mail: basili@fc-md.umd.edu

Kathleen Dangle is an FC-MD division director where she works with organizations to implement software management-related improvements such as software measurement, acquisition, and Capability Maturity Model® Integration-based processes.

E-mail: kdangle@fc-md.umd.edu

Linda Esker is an FC-MD senior engineer, providing expertise to government programs in program management and software development as well as metrics definition and analysis.

Phone: (301) 403-8967
E-mail: lesker@fc-md.umd.edu

Frank Marotta is a mathematician at the U.S. Army Aberdeen Test Center and has more than 20 years experience in software testing of Army weapon systems, test optimization, and software metrics.

E-mail: frank.marotta@us.army.mil

Ioana Rus is a member of the Honeywell Aerospace Software Center for Excellence. She works in process modeling and simulation, empirical studies, software dependability, and measurement.

E-mail: irus@computer.org

® Capability Maturity Model is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Safety and Security: Certification Issues and Technologies[1]

Dr. Benjamin M. Brosgol
AdaCore

Many military systems are safety-critical, with failure possibly resulting in loss of human life. In today's interconnected environment, safety requires security. Compliance with the higher levels of safety or security standards demands a disciplined development process along with appropriate programming language and toolset technology. Since full, general-purpose languages are too large and complex to be usable for safety-critical or high-security systems, a key requirement is to define subsets that are simple enough to facilitate certification but expressive enough to program the needed application functionality. This article summarizes representative safety and security standards (DO-178B and the Common Criteria, respectively), identifies the language-related issues surrounding safety and security certification, and assesses three candidate technologies—C (including C++), Ada (including SPARK), and Java—with respect to suitability for safety-critical or high-security systems.

Compliance with formal safety standards is becoming a major consideration, and sometimes a requirement, in defense systems. The safety standard that is most relevant is DO-178B [1], which is used by the Federal Aviation Administration (FAA) for the certification of commercial aircraft. DO-178B is process-oriented; certification is not so much a safety assessment of the completed system, but rather an evaluation of a set of developer-provided artifacts. These artifacts are intended to provide both direct evidence of sound software engineering practice and lend assurance (through indirect evidence) of the safety of the resulting system.

DO-178B identifies five levels of criticality: A (highest) through E (lowest). Table 1 characterizes the various levels and identifies the number of DO-178B requirements that apply. The term *with independence* means that the evidence for meeting a requirement must be supplied by someone other than the developer. The term *safety-critical* generally applies to software at levels A and B, which demand greater rigor and more comprehensive analysis than the lower levels.

DO-178B focuses on requirements-based testing and bi-directional traceability (from requirements to code, and vice versa) as key elements of software verification. Test cases must fully cover the code; *dead code* (unexercised code that does not correspond to a specific requirement) must be removed.

DO-178B is open to criticism on several grounds:

- It does not directly assess the safety of the resulting system.
- Although the artifacts are process-oriented, there is no guarantee that sound processes were followed and it is not rare for developers to prepare the DO-178B artifacts for previously developed components *a posteriori*.
- Its emphasis on testing does not adequately take into account alternative technologies (such as formal methods) for providing safety assurance.
- It is unclear on how its objectives apply to modern software development approaches such as object-oriented technology (OOT).

These problems should not be overemphasized. DO-178B has been successful in practice: Although there have been some close calls, DO-178B-certified software has never been the direct cause of an aircraft accident resulting in a fatality. However, much has changed in the software industry since the early 1990s when DO-178B was written. Work is in progress on a successor, DO-178C [2], that will attempt to address some of the perceived issues with DO-178B. For example, DO-178C will accommodate newer software technologies (OOT, model-based design) and alternative software verification techniques (formal methods, abstract interpretation).

Security Certification

Security is generally defined as the protection of assets against threats to their *confidentiality*, *integrity*, and/or *availability*. Designing an information technology (IT) product for security thus involves design steps (avoiding vulnerabilities that adversaries could exploit to compromise these requirements) as well as runtime actions (detecting and responding to attempted breaches).

DO-178B says nothing explicit about security, but a system with security vulnerabilities is at risk for exploitation by adversaries to render it unsafe. The *safety requires security* principle has two implications. First, an organization developing safety-critical software should adopt methodologies and design frameworks that can help realize security requirements. Guidelines such as the "Defense Acquisition Guidebook" [3] and architectures such as multiple independent levels of security (MILS) [4] are relevant. Second, it must be possible to assess whether a product with safety-critical requirements is sufficiently secure. The Common Criteria/Common Evaluation Methodology [5] provides such an assessment mechanism. These international standards include a catalog of security-functional and assurance requirements and a process for evaluating the security characteristics of a given IT product.

Somewhat analogous to the levels of DO-178B, the Common Criteria defines seven Evaluation Assurance Levels (EALs), numbered from 1 to 7 in increasing order of criticality. Generally speaking, EAL 4 corresponds to best commercial practice without a serious focus on security threats; higher levels require special security-oriented mechanisms and increased

Table 1: *Criticality Levels in DO-178B*

Level	Condition	Effect of Anomalous Behavior	Number of Objectives
A	Catastrophic failure	"... prevent continued safe flight and landing..."	66 (14 *with independence*)
B	Hazardous/severe failure	"... serious or potentially fatal injuries to a small number of … occupants ..."	65 (11 *with independence)*
C	Major failure	"... discomfort to occupants, possibly including injury..."	57
D	Minor failure	"... some inconvenience to occupants..."	28
E	None	"... no effect on aircraft operational capability or pilot workload..."	0

effort in demonstrating compliance. At EAL 7, formal methods (i.e., mathematics-based analyses) are required to demonstrate that security requirements are met.

With safety implying the need for security, it is reasonable to consider certifying a system against standards for both. This idea is not new; the SafSec project [6], sponsored by the United Kingdom's Ministry of Defense, has developed an integrated methodology for dual safety and security certification for avionics. In another effort, a group at the University of Idaho has analyzed the correspondence between DO-178B and the Common Criteria, mapping DO-178B objectives to Common Criteria elements [7], and has studied the feasibility of joint certification. However, the practicality of applying the Common Criteria to large Department of Defense (DoD) systems is unclear. As summarized in a 2007 Report of the Defense Science Board Task Force:

> Criticisms of Common Criteria-based schemes are that they are expensive, require artifacts that are not produced until well after product design and implementation, do not substantially reduce implementation-level vulnerabilities when using today's software development practices, and lack thorough penetration analysis at EAL 4 and below. [8]

Furthermore, the fact that a product has been certified at a specific EAL means very little by itself. First, it says nothing about the quality of the product outside the security functional requirements. Second, a prospective consumer needs to understand which of these requirements are being implemented and whether the vendor-assumed operational environment (the severity of the threats/assumed skills of the adversaries) matches reality.

Notwithstanding how it is assessed, security is obviously necessary for safety, and safety certification agencies are paying increasing attention to the relationship between the two. As an example, an FAA "Special Conditions" notice directed one supplier to demonstrate the independence of the networks for passenger-accessible components and flight control on one of its aircraft [9].

A Comparison of Safety and Security Certification Issues

DO-178B and the Common Criteria have some basic similarities:

- Concern with the full software development life cycle—including peripheral activities such as configuration management—in an attempt to catch human (developer) error before the system is fielded.
- Tiered approach (criticality levels) reflecting real-life trade-offs: Resources are finite, and a system must be safe/secure enough for its intended purpose.
- Emphasis on testing as a major element of software verification.

There are also some important differences:

- **Scope of requirements.** DO-178B deals with the entire system; the Common Criteria focuses almost exclusively on just the security functional requirements.
- **Functional requirements.** There is no specific set of safety functions called out in DO-178B. In contrast, the security domain has well-defined functional requirements that need to be implemented.
- **System users/operators.** An IT product must be immune to attacks from unknown and possibly malevolent users who can directly supply input. Input to a safety-critical system is generally supplied by known operators whose trustworthiness has been separately vetted.

In one sense, compliance with safety standards is more demanding:

- Each component must be certified against requirements for its safety level.
- At the higher levels, it is necessary to both demonstrate the absence of dead code and perform structural testing to verify the absence of non-required functionality.
- For EAL compliance, the specific development and testing requirements apply only to the security functions and not to the entire IT product; there is no prohibition against dead code/extra functionality (although such code must be shown to be free from vulnerabilities).

In another sense, compliance with security standards is more demanding:

- Formal methods are required at EAL 7.
- Vulnerability analysis is difficult and must assume a sophisticated and malevolent adversary; for safety, the adversaries are the laws of physics.

Although safety requires security, the relationship in the other direction is not so immediate. Most IT products for which security is critical do not control systems where life is at stake and, thus, safety is generally not an issue.

In the context of overall system design, safety and security sometimes conflict, especially with respect to behavior under failure conditions. Taking a system offline to protect data may be reasonable behavior for security, but if the data are needed for flight control/management, then such a policy may have disastrous consequences for safety. *Fail-safe* is not the same thing as *fail-secure*. These sorts of conflicts need to be resolved during design, with appropriate trade-offs based on the anticipated risks.

Programming Language Requirements

The programming language choice is arguably the most important technical decision that the developer organization will make. As summarized in a National Academy of Sciences report:

> The overwhelming majority of security vulnerabilities reported in software products—and exploited to attack the users of such products—are at the implementation level. The prevalence of code-related problems, however, is a direct consequence of higher-level decisions to use programming languages, design methods, and libraries that admit these problems. [10]

Although directed at security issues, these comments apply equally to safety. The programming language plays a key role in determining the ease or difficulty of developing software that avoids vulnerabilities and that is certifiable against safety or security standards.

A simple example of a programming language feature that can easily lead to an application vulnerability is the C library function *gets()*, which reads a character string as input from a user until an *end-of-line* or *end-of-file* is encountered. The program can only pre-allocate an area of some fixed length as the destination, but a user can accidentally or intentionally supply input that exceeds this bound. The effect is the classical *buffer overflow*, in which the excess characters overwrite other data, possibly including a function's return address. By crafting an input string with specific content, a malevolent user can take control of the machine to execute arbitrary code.

Although DO-178B and the Common Criteria do not offer direct guidance on a programming language choice, it is possible to abstract from their specific objectives and infer several general requirements that a language must meet. The following sections discuss four of these requirements: reliability, predictability, analyzability, and expressiveness.

Reliability

The language should promote the development of readable, correct code as well as:

- Have an intuitive lexical and syntactic structure and be free of *traps* and *pitfalls*.
- Help detect errors early (at compile time if possible), and should prevent errors such as out-of-range array indices and references to uninitialized data.
- Help (if it supports concurrent programming with *threads/tasks*) in avoiding errors such as unprotected accesses to shared data, race conditions (where the effect of the program depends on the relative speed of the threads/tasks), and deadlock.

Predictability

The language specification should be unambiguous. Implementation-dependent or, worse, undefined behavior introduces vulnerabilities because the effect of the program may not be as the developer had intended.

Analyzability

The language should facilitate both static analysis (detecting uninitialized variables, identifying dead code, predicting maximum stack usage and worst-case execution time, etc.) and dynamic analysis (requirements-based or structure-based testing). A useful catalog of such analysis techniques (from [11]) is given in Table 2.

The various analysis techniques impose constraints on the programming language. For example, control and data flow analyses generally prohibit the use of *goto* statements, stack usage analysis generally prohibits recursion, and coverage analysis may preclude the use of source code constructs that generate implicit loops or conditionals. As a result, there is no such thing as *the* safety-critical or high-security subset of a given language. The particular subset used either determines or is determined by the analysis techniques that will assist in demonstrating compliance with the operative certification standard.

Automated static analysis tools play an important role in the safety and security domains; indeed, there is a U.S. Department of Homeland Security-sponsored project under way—Static Analysis Metrics and Tool Evaluation (SAMATE) [12]—identifying and measuring the effectiveness of such tools. Static analysis tools are most successful during program development, where they can help detect problems before they occur, versus as a technique for detecting vulnerabilities *a posteriori* in existing code.

Approach	Group Name	Technique(s)
Static Analysis	Flow Analysis	Control Flow
		Data Flow
		Information Flow
	Symbolic Analysis	Symbolic Execution
		Formal Code Verification
	Range Checking	Range Checking
	Stack Usage	Stack Usage
	Timing Analysis	Timing Analysis
	Other Memory Usage	Other Memory Usage
	Object Code Analysis	Object Code Analysis
Dynamic Analysis (Testing)	Requirements-Based Testing	Equivalence Class
		Boundary Value
	Structure-Based Testing	Statement Coverage
		Branch Coverage
		Modified Condition/ Decision Coverage

Table 2: *Analysis Techniques*

Expressiveness

The language should support general-purpose programming, either through language features or auxiliary libraries, and should also offer specialized functionality (as required). For real-time safety-critical systems, this means support for interrupt handling, low-level programming, concurrency, perhaps fixed-point arithmetic, and other features. For high-security systems, it means mechanisms are needed for implementing security-functional requirements (e.g., for cryptography).

Unfortunately, language generality (as implied by the expressiveness requirement) directly conflicts with the analyzability requirement. That conflict complicates the language selection decision.

Object-Oriented Technology

The history of programming languages has seen a steady evolution of features that promote maintainable software and many of these features directly support the reliability and analyzability requirements previously described. However, some of the advances present difficulties for safety and security certification. Perhaps the most significant example is OOT [13], found in such languages as C++, Ada 95, and Java. OOT is not addressed in DO-178B, and there are indeed a number of challenges:

- **A paradigm clash.** OOT's distribution of functionality across classes conflicts with DO-178B's focus on tracing between requirements and implemented functions.
- **Technical issues.** The features that are the essence of OOT complicate safety and security certification. For example, dynamic binding typically is implemented by a compiler-generated data structure known as a *vtable* (a table of addresses of functions). For safety or security certification, the developer must demonstrate that the *vtable* is properly initialized and that it cannot be corrupted.
- **Cultural issues.** Certification authority personnel are not necessarily language experts and may (rightfully) be concerned about how to deal with unfamiliar technology.

A series of workshops several years ago produced a handbook [14] that addressed these issues in detail. The in-progress work on DO-178C is taking these into account, and it is likely that the eventual new standard will offer some direct guidance in connection with OOT. However, developers are not waiting for DO-178C. OOT is currently being used in safety-critical code; as one example, an avionics system using Ada 95's object-oriented features has been certified at Level A. It seems inevitable—as experience with OOT and certification is gained—that usage of object-oriented languages will increase.

Candidate Programming Languages

Although (in principle) any programming language could be used for developing safety-critical or high-security software, the requirements for reliability, predictability, and especially analyzability imply that suitable subsets be chosen. The key issues are how a language can be subsetted to ease certification for applications restricted to the subset, and whether the language has intrinsic problems that cannot be removed by subsetting.

This section summarizes how several current language technologies—either

currently in use or under consideration for safety-critical systems—compare with respect to subsettability.

C-Based Technology[2]

MISRA-C

The United Kingdom-based Motor Industry Software Reliability Association (MISRA) has produced a set of language restrictions, called MISRA-C [16], which attempts to mitigate C's vulnerabilities. MISRA-C codifies best practices for C programming and has become somewhat of a *de facto* standard as a C subset for critical systems. Benefits stem from C's relative simplicity, the large population of C programmers, and a wide assortment of tools and service providers. MISRA-C has been used successfully in safety-critical systems.

On the other hand, MISRA-C has some significant drawbacks:

- C was not designed for safety-critical systems, and some intrinsic issues (e.g., the wraparound semantics for integer overflow) cannot be removed by subsetting.
- Despite MISRA-C's stated goals, the rules are not always enforceable by static tools, and different tools may enforce the subset differently.
- Since concurrency is not provided in C (it is only available through external libraries), MISRA-C offers no guidance on how to use C in a multithreaded environment.

C++

C++ [17] is in many ways a better C. Developers of safety-critical systems often have staff knowledgeable in C++ and may possess existing C++ components that they would like to re-use in a certified system.

To help meet this goal, coding standards such as Joint Strike Fighter C++ [18], and MISRA C++ [19] have been developed. These extend or adapt MISRA-C to deal with C++'s additional facilities. The rules constrain the usage of language features in order to avoid problems and to promote good style.

Safe C++ coding standards are essential if C++ is chosen, and C++ has been used to develop safety-critical systems. However, the previously noted drawbacks for MISRA-C apply here, and the OOT coding guidelines do not solve the underlying certification issues.

Ada-Based Technology

Ada

Ada [20] was designed to be used for safety-critical systems. It avoids many of the C and C++ vulnerabilities (e.g., checking for out-of-range array indexing and integer overflow), and also offers a standard set of concurrent programming features. Ada continues being widely used for safety-critical systems including military and commercial avionics.

Full Ada is too large to be practical for safety certification so subsetting is required. Ada provides a unique approach to this issue, allowing the application to specify the features that are to be excluded. This means no runtime support libraries for such features, and compile-time error detection of attempted uses. The *à la carte* style to defining language subsets is flexible and does not require specialized tool support: a standard compiler performs the necessary analysis.

The latest Ada language standard also includes the Ravenscar profile [21], a certifiable subset of concurrency features.

> *"Ada continues being widely used for safety-critical systems including military and commercial avionics."*

Ada's disadvantages for safety-critical systems are largely external (non-technical). Ada usage is not as widespread as other languages and, thus, its tool vendor community is smaller. On the technical side, Ada does not directly address vulnerabilities such as references to uninitialized variables. As with C and C++, supplementary analysis is required to detect/prevent such errors.

SPARK

SPARK [22] is a subset of Ada 95, augmented by specially formed comments known as *contracts* (or *annotations*), designed to facilitate a rigorous, static demonstration of program correctness. SPARK omits features that complicate analysis or formal proofs or that interfere with bounded time/space predictability. The language includes most of Ada 95's static features as well as the Ravenscar concurrency profile, and the semantics are completely unambiguous (no implementation-dependent or undefined behavior).

Contracts in SPARK specify data and information flow, inter-module dependencies, and dynamic invariants (pre-/post-conditions, assertions). The SPARK tools analyze the program to ensure that the code is consistent with the contracts and that no runtime exceptions will be raised. They detect errors such as potential references to uninitialized variables and dead code. The static analysis performed by the SPARK tools is *sound* (there are no *false negatives*, an especially important requirement in connection with safety certification) with a low false alarm rate (there are few *false positives*). The SPARK tools can also generate verification conditions and automate the proof of these conditions.

SPARK has been used in practice on a variety of systems, both safety-critical and high-security. Of all the candidate language technologies, SPARK best meets the requirements for reliability, predictability, and analyzability. Its main technical drawback is with expressibility, as it has a rather restricted feature set. Additionally, the SPARK infrastructure (user/vendor community) is smaller than that of other language technologies.

Java-Based Technology

Java [23] seems simultaneously logical and curious as a technology choice for safety-critical systems. On one hand, it was designed with careful attention to security: Its initial goal was to enable downloadable *applets* to be executed on client machines without risk of compromising the confidentiality or integrity of client resources. The Java language is largely free from the implementation dependencies found in C, C++, and Ada, such as order of expression evaluation. Java also performs conservative checks to prevent unreachable (dead) code and references to uninitialized variables. It provides automatic storage management (*garbage collection*) instead of an explicit *free* construct that is the source of subtle errors in other languages.

Security, however, is not the same as safety. Indeed, Java technology has limitations for safety-critical systems, falling into two general categories:

1. Ensuring real-time predictability.
2. Meeting certification standards such as DO-178B.

Both of these have been the subject of Java Specification Requests (JSRs) under Sun Microsystems' Java Community Process [24]. JSRs 1 [25] and 282 [26] have defined the Real-Time Specification for Java (RTSJ); JSR-302 [27], in progress, is defining a subset of the RTSJ that is intended for Java applications that need to be certified to DO-178B at levels up to A.

The RTSJ extends the Java platform to add real-time predictability. The main enhancements are for concurrency (to define scheduling semantics more precisely than in standard Java, and to prevent pri-

ority anomalies) and memory management (to avoid garbage collection interference).

The RTSJ, as an extension of the standard Java platform, is not appropriate (and was not intended) for safety-critical applications. It is too complex and some of its major features (especially in the memory management area) require runtime checks that may be expensive. However, it does address Java's real-time issues and, thus, is serving as the basis for JSR 302's safety-critical Java specification. This in-progress effort defines three levels of support for safety-critical systems: (1) a traditional cyclic executive (no threading); (2) a thread-based approach with simple memory management; and (3) a thread-based approach with more general memory management. Each is characterized by a corresponding subset of RTSJ functionality and Java class libraries. JSR-302 exploits Java 1.5's annotation feature: A developer annotates various properties of the code (for example, memory usage), and static analysis tools verify the annotations' correctness.

Of all the language technologies that are candidates for safety-critical development, Java has the most significant challenges:

- The Virtual Machine execution environment for Java programs is unconventional, blurring the distinction between code and data and raising safety certification issues.
- Unlike C++ and Ada (where OOT is available but optional), Java is based around object orientation. It is possible to use Java without taking advantage of OOT, but the style is contrived.
- Java is lexically and syntactically based on C, therefore sharing a number of that language's traps and pitfalls.
- The RTSJ/JSR-302 approach to memory management is rather complicated, and requires Java programmers to carefully analyze dynamic memory usage.

Despite these issues, there is interest in safety-critical Java from both the developer and the user communities. An organization that has adopted Java as an implementation language on a project may have some components with safety-critical requirements, and keeping the entire system within one language can simplify some aspects of project management.

Conclusion

Developing safety-critical/high-security systems is difficult. The key skill is not so much the knowledge of a particular programming language; a software professional should be able to learn a new language in a short amount of time. The more critical talent is the ability to think through a design and implementation with a focus not just on meeting a system's functional requirements but also on avoiding hazards and vulnerabilities. Such *negative programming*—ensuring that bad things do not happen—requires careful analysis and a defensive development approach that, in turn, places demands on the programming language and tools. For software safety and security, the idea of *minding your language* is more than a matter of etiquette; it could be the key to a system's success.◆

References

1. RTCA SC-167/EUROCAE WG-12. RTCA/DO-178B. Software Considerations in Airborne Systems and Equipment Certification. Dec. 1992.
2. Embry-Riddle Aeronautical University. "Software Considerations in Airborne Systems." <http://forum.pr.erau.edu/SCAS>.
3. Defense Acquisition University. Defense Acquisition Guidebook. 20 Dec. 2004 <http://akss.dau.mil/dag/>.
4. Vanfleet, W. Mark, et al. "MILS: Architecture for High-Assurance Embedded Computing." CrossTalk Aug. 2005 <www.stsc.hill.af.mil/crosstalk/2005/08/0508Vanfleet_etal.html>.
5. The Common Criteria Portal. Common Criteria for Information Technology Security Evaluation. Vers. 3.1. Sept. 2006 <www.commoncriteriaportal.org/thecc.html>.
6. United Kingdom Ministry of Defense and Praxis High Integrity Systems. SafSec Methodology: Standard. 2 Nov. 2006 <www.praxis-cs.com/safsec/downloads/SafSec_Methodology_Standard_Material_pdf.pdf>.
7. Taylor, Carol, Jim Alves-Foss, and Bob Rinker. "Merging Safety and Assurance: The Process of Dual Certification for Software." STC 2002 <www.csds.uidaho.edu/papers/Taylor02d.pdf>.
8. Defense Science Board. Report of the Defense Science Board Task Force on Mission Impact of Foreign Influence on DoD Software. Washington: Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics. Sept. 2007: 33.
9. Department of Transportation, FAA. "Special Conditions: Boeing Model 787-8 Airplane; Systems and Data Networks Security – Isolation or Protection From Unauthorized Passenger Domain Systems Access." Federal Register. 2 Jan. 2008.
10. Jackson, Daniel, Martyn Thomas, and Lynette I. Millett, Eds. Software for Dependable Systems: Sufficient Evidence? Washington: National Academies Press. 27 Aug. 2007 <http://books.nap.edu/catalog.php?record_id=11923>.
11. International Organization for Standardization. ISO/IEC TR 15942, Guide for the Use of Ada in High-Integrity Systems. 2000 <http://std.dkuug.dk/JTC1/SC22/WG9/n359.pdf>.
12. National Institute of Standards and Technology. SAMATE – Software Assurance Metrics and Tool Evaluation. July 2005 <http://samate.nist.gov>.
13. Meyer, Bertrand. Object-Oriented Software Construction. 2nd ed. Santa Barbara, CA: Prentice Hall PTR, 1997.
14. FAA. Handbook for Object-Oriented Technology in Aviation. 26 Oct. 2004 <www.faa.gov/aircraft/air_cert/design_approvals/air_software/oot>.
15. International Organization for Standardization. ISO/IEC 9899: 1990; Programming Languages - C.
16. MISRA-C: 2004. Guidelines For the Use of the C Language in Critical Systems. Nuneaton, U.K.: MISRA Ltd., 2004.
17. International Organization for Standardization. ISO/IEC 14882: 2003; Programming Languages – C++.
18. Lockheed Martin Corp. Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program. Dec. 2005 <www.jsf.mil/downloads/documents/JSF_AV_C%2B%2B_Coding_Standards_Rev_C.doc>.
19. MISRA-C++: 2008. Guidelines For the Use of the C Language in Critical Systems. Nuneaton, U.K.: MISRA Ltd., 2004.
20. International Organization for Standardization. ISO/IEC 8652:1995/Amd 1: 2007 Programming Languages - Ada.
21. Dobbing, Brian, and Alan Burns. "The Ravenscar Profile for Real-Time and High Integrity Systems." CrossTalk Nov. 2003 <www.stsc.hill.af.mil/crosstalk/2003/11/0311Dobbing.html>.
22. Barnes, John. High Integrity Software – The SPARK Approach to Safety and Security. Addison-Wesley, 2003.
23. Arnold, Ken, James Gosling, and David Holmes. The Java Programming Language. 4th ed. Addison Wesley, 2006.
24. Community Development of Java Technology Specifications. The Java Community Process℠ Program. 2008 <www.jcp.org/en/home/index>.

Coming Events

December 1-3
Defense Network Centric Operations
Arlington, VA
www.wbresearch.com/DNCO

December 1-4
26th Army Science Conference
Orlando, FL
www.asc2008.com

December 1-4
Interservice/Industry Training, Simulation, and Education Conference
Orlando, FL
www.iitsec.org

December 7-10
Winter Simulation Conference
Miami, FL
http://wintersim.org

November 16-18
Software Engineering and Applications
Orlando, FL
www.iasted.org/conferences/home-632.html

November 18-20
SpecOps East 2008
Fayetteville, NC
www.defensetradeshows.com/SPECOPSEAST08_General_Info.html

November 18-22
Joint International Conference on Engineering and Technology
Nashville, TN
www.ijme.us/IJME_Conference_2008/

April 20-23, 2009
21st Annual Systems and Software Technology Conference
Salt Lake City, UT
www.sstc-online.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**

25. Community Development of Java Technology Specifications. JSR 1: RTSJ. 12 July 2006 <www.jcp.org/en/jsr/detail?id=1>.
26. Community Development of Java Technology Specifications. JSR 282: RTSJ. 12 Sept. 2005 <www.jcp.org/en/jsr/detail?id=282>.
27. Community Development of Java Technology Specifications. JSR 302: Safety Critical Java™ Technology. 24 July 2006 <www.jcp.org/en/jsr/detail?id=_302>.

Notes

1. This article is based on a tutorial, "Safety and Security: An Analysis of Certification Issues and Technologies," presented by the author at the Systems and Software Technology Conference, 2008.
2. In this section, C means the 1990 version of the ISO language standard [15].

About the Author

Benjamin M. Brosgol, Ph.D., is a senior member of the technical staff at AdaCore. He has been actively involved with the Ada language effort since its outset and has received the SIGAda Outstanding Ada Community Contributions award. Brosgol has a bachelor's degree in mathematics from Amherst College, as well as master's and doctorate degrees in applied mathematics from Harvard University.

AdaCore
104 Fifth AVE 15th FL
New York, NY 10011
Phone: (212) 620-7300
E-mail: brosgol@adacore.com

Letter to the Editor

Dear CrossTalk Editor,

Reading August's 20th anniversary edition—especially Gary Petersen's CrossTalk*: The Long and Winding Road*—triggered my own early memories of the journal and the Software Technical Support Center (STSC). I have always appreciated CrossTalk, from the authors' real-world application of concepts to the "above-and-beyond" assistance of your staff. We at Northrop Grumman put what we learned into practice.

In 1989, our group at Northrop (before adding the Grumman) utilized your articles on the Department of Defense's (DoD) requirements for the Capability Maturity Model Integration (CMMI®) Level 3. While I cannot remember the authors or titles, these articles aided our logistics engineers in developing a quality approach as we started our transformation into what is now much-renowned Software Engineering Processing Group.

In the mid-90s, Watts S. Humphrey—who was, not surprisingly, part of the 20th anniversary issue—was one of several CrossTalk authors whose articles pointed the way toward DoD systems management of large-scale software and systems engineering integration. As well, articles on software project management were one of the tools used to kick off Northrop's CMMI Level 3 effort and organize a more integrated project management approach to B-2 software engineering.

Around this same time, the editorial team of CrossTalk invited Northrop personnel to participate in STSC meetings and presentations, and introduced us to senior DoD system managers. These sessions were the genesis of our software engineering and systems engineering breakthroughs. And, although it may seem like a small gesture, supplying us with the proceedings from these gatherings helped educate and motivate our teams of software engineers and supported our argument that we needed new hardware and better computer-aided software engineering tools.

We saved more than 100,000 man-hours by implementing methodologies gleaned from CrossTalk, equating to approximately $8.3 million per year (in 1993 dollars) or $33 million over the four-year period of development. This is quite a savings when compared to our estimated $121 million annual budget. We've all received plentiful kudos for our achievements, but we would like to pass along some of that gratitude to CrossTalk's authors and staff.

—John B. Burger
Northrop Grumman (retired)
4940 Flora Vista LN
Sacramento, CA 95822
<jjburger@aol.com>

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

WebBee: A Platform for Secure Mobile Coordination and Communication in Crisis Scenarios

Sugih Jamin
University of Michigan[1]

Recently, disaster scenarios and terrorist attacks have made apparent some fundamental shortcomings in first responders' conventional coordination infrastructures. For example, unsatisfactory device connectivity and security vulnerabilities made evident by devices' inherently mobile nature have the potential to seriously compromise first responders' effectiveness. To address these shortcomings, our team designed and built WebBee, a secure coordination and communication infrastructure. This article will take a high-level look at WebBee's architecture, and examine some interesting, non-trivial sample applications we have deployed on top of it.

Ever since the September 11, 2001 terrorist attacks, the United States has been re-evaluating coordination for first responders in disaster scenarios. First responders must communicate reliably and securely in times of crisis. However, communication channels such as cell phone networks may be impaired or destroyed during disaster scenarios. Even if communication was *technically* feasible through these channels, extreme congestion might render them useless for first responders. Another problem is that these channels are more vulnerable to compromise: A malicious agent could steal a first responder's cell phone and intercept communications. This can seriously undermine a first responder's effectiveness in crisis situations.

The first responders have three primary needs. They must be able to communicate using devices they likely already have and are well-accustomed with. Secondly, the communication channel must be secure in mobile environments. Finally, while in a time of crisis, the consumer communication infrastructure can sometimes be used, it cannot be relied upon solely. WebBee addresses each of these concerns.

Architecture

There are three major components of the WebBee architecture (as shown in Figure 1 on the following page): the *instant infrastructure*, the *WebBee coordination server*, and the *database server*. The system has been designed so that components can be distributed across different machines.

Certain field personnel are equipped with battery-operated instant infrastructure backpack units. Equipment is commercial off-the-shelf hardware, so very large numbers of personnel can be outfitted easily. Custom SMesh software [1] helps maximize connectivity by dynamically reorganizing the network topology as personnel move about the field. The WebBee coordination server is an abstraction of several components that coordinate request handling, challenge-response management, policy examination, application hosting, and message dispatching. The database server manages all data interactions.

WebBee Coordination Server Component Detail

WebBee Master Server and Challenge Server Interaction

The WebBee master server negotiates traffic from clients between the challenge server and the application bridge. When a client request comes in, the WebBee master server stores it and asks the challenge server whether the client needs to be challenged. If the challenge server determines no challenge is needed, it tells the WebBee master server that it is OK to proceed. Otherwise, the challenge server issues a challenge through the master server to the client. The client's solution is sent back through the master server to the challenge server. If it is invalid, the challenge server informs the master server that no action is to be taken and the client is informed that the request was denied. If the solution is valid, the WebBee master server retrieves the client's most recent request and dispatches it to the application bridge. Our model, therefore, assumes that clients will only ever need a single request serviced at a time.

"... [communication] channels are more vulnerable to a compromise: a malicious agent could steal a first responder's cell phone and intercept communications. This can seriously undermine a first responder's effectiveness ..."

Security

Our security mechanism is broken into three separate subsystems: the challenge server, upload security, and download security. All are wrapped in a secure sockets layer.

The Challenge Server

The challenge server's job consists of *policies* and *challenges*. Policies encode conditions under which challenges are required, and are arranged in a hierarchy: If an agent passes one policy, there may still be subsequent policies that must be evaluated. The policy scheme for the WebBee coordination server is depicted in Figure 2 (see next page).

The first policy here is an *application-level* test. This special policy grants full access to certain applications, and demonstrates that WebBee supports both secure and non-secure applications. If the application must be challenged, a *temporal* policy is activated to determine if the client's last challenge-response has expired. If it has expired, the client is issued a challenge. The last policy is a *geospatial* policy: If the user has strayed far away from the set of last known global positioning system (GPS) coordinates, the client is challenged.

Policy intervals can be defined on a per-user basis, based on the level of security required for each client. At most, one challenge will occur through a traversal of this policy flowchart. Once the client solves the challenge, his or her GPS coor-

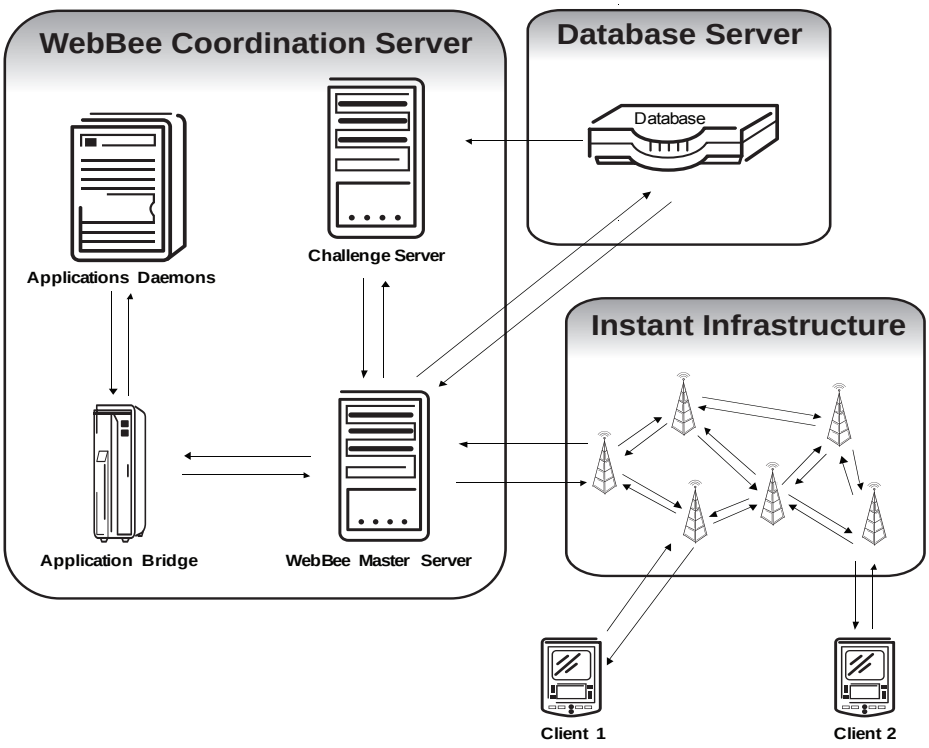


Figure 1: *WebBee Component Architecture*

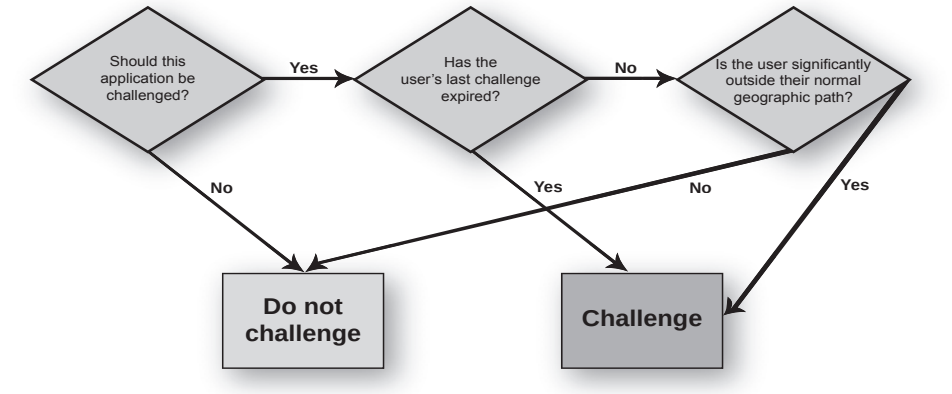


Figure 2: *Policy Flowchart for the WebBee Coordination Server*

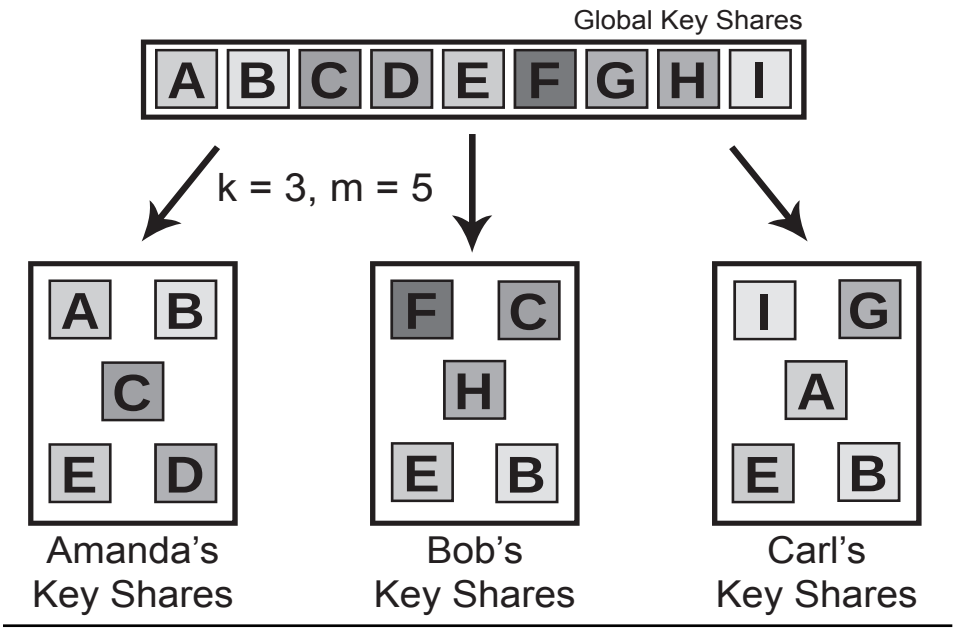


Figure 3: *Amanda, Bob, and Carl Initially All Have Valid Keyshares*

dinates and a timestamp are stored in the database.

When the policy flowchart determines that a challenge is required, the server randomly selects one of several possible challenges and issues it to the client. If the client solves it, then the request is serviced. Otherwise, the current and all subsequent requests will also be denied until the client successfully solves the original challenge. This eliminates malicious clients' ability to *game the system* by exploring the challenge space.

Currently, only text-based (e.g., password) challenges have been implemented. With the right hardware, the challenge system could be extended to issue other kinds of challenges, such as biometric challenges (e.g., fingerprint, voice, and/or retinal scanning).

Upload Security

In our scalable crisis management system, we are assuming that there are many downloads but relatively few uploads. With this in mind, we have decomposed our security requirements into upload and download security characteristics.

For upload security, if a handheld is lost, we want to ensure that (1) data that has already been posted cannot be repudiated, and (2) data cannot be post-dated. Our forward secure signatures use a private key that evolves as a function of time; the public key, however, remains the same. This kind of forward-secure scheme was proposed by Anderson [2] and implemented by Bellare, Mihir, and Miner [3].

Download Security: The Quorum System

For download security, scaling is an important issue. For clients, we want to require relatively few of their staff to have to acquire new keys during a change (e.g., departure or loss of device). The quorum system implements download security with these kinds of scalability concerns in mind.

In the quorum system, agents need to have a minimum number, k, of *keyshares* to securely read a message. At initialization, each agent receives m keyshares, where $m > k$, from a global keyshare set consisting of a total of s keyshares. If a user leaves, his or her shares are invalidated for *all* users. When a user has fewer than k valid shares, they must obtain a new set of valid keyshares from the global keyshare collection.

When the server broadcasts a message, it first encrypts it under a *message key*. This key, in turn, is itself encrypted s times. The s-encrypted message keys and

the encrypted message are sent to all agents who decrypt the message keys using their personal keysets. If exactly *k* of the keys are identical, it is valid and the agent proceeds to decrypt the encrypted message with that decrypted message key.

Figures 3, 4, and 5 depict a scenario in which $k = 3$ and $m = 5$. In Figure 3, Amanda, Bob, and Carl all have a quorum of valid keyshares. In Figure 4, when Bob leaves, three of Amanda's keyshares are invalidated, forcing her to obtain new shares. Carl only has two shares invalidated; he can continue to operate. Figure 5 depicts the scenario in which Amanda has reported a lost or stolen handheld, in which case all of Amanda's keyshares are invalidated. In this instance, Carl must reacquire new keyshares to operate.

Application Bridge

The application bridge dispatches requests to the appropriate application daemon via an ID embedded in the request header. If a response is generated, it is sent back through the WebBee master server to the client. Gas Prices, Event Reports, and Agent Contingency and Action Coordinator (AC2) are three applications we have built using the WebBee framework.

Gas Prices

The Gas Prices application allows clients to determine the gas stations with the cheapest prices. A client initially sends a request containing his or her GPS coordinates. The Gas Prices daemon constructs a map through an implementation of the U.S. Census Bureau's Topologically Integrated Geographic Encoding and Referencing (TIGER) geographic information system (GIS) database [4], then queries a Web site that publishes up-to-date gas prices and sends it back to the client.

Gas Prices and other applications use the WebBee scraping engine to obtain data from the Web. For each application, a *scraping script* identifies the data components of interest in a Web page. Any static or dynamic data can be acquired—including text, images, and audio.

Event Reports

The Event Reports application (see Figure 6, next page) allows clients to log incidents that they observe in the field. Other clients are notified about these incidents only once they become geospatially relevant. Clients specify details about an incident by typing out a short message—as well as a radius in meters—on the handheld device. As other clients move in range, their handhelds are notified via the short messaging service (SMS). This relieves clients of having to sift through reports to determine which are immediately important, enabling him or her to react faster and more effectively.

A scenario is shown in Figure 7 (see next page). A report about a fire at the Chicago Mercantile Exchange (A) is submitted. One fire department unit (B) and two police department units (C) and (F) receive the alert about the fire. Another report about an unrelated incident is submitted by an informant across the city (D). Here, one fire department unit is alerted (E), as is one police department unit (F). Notice that (F) receives alerts about both incidents since it is in range of both. By contrast, another police department (G) receives no alerts. As soon as G moves into range (if ever), they will receive the report.

Event Reports – Exploiting Database Triggers for Better Performance

Report notifications to clients are implemented through database triggers. The WebBee database server contains an

Figure 4: *Bob Leaves*

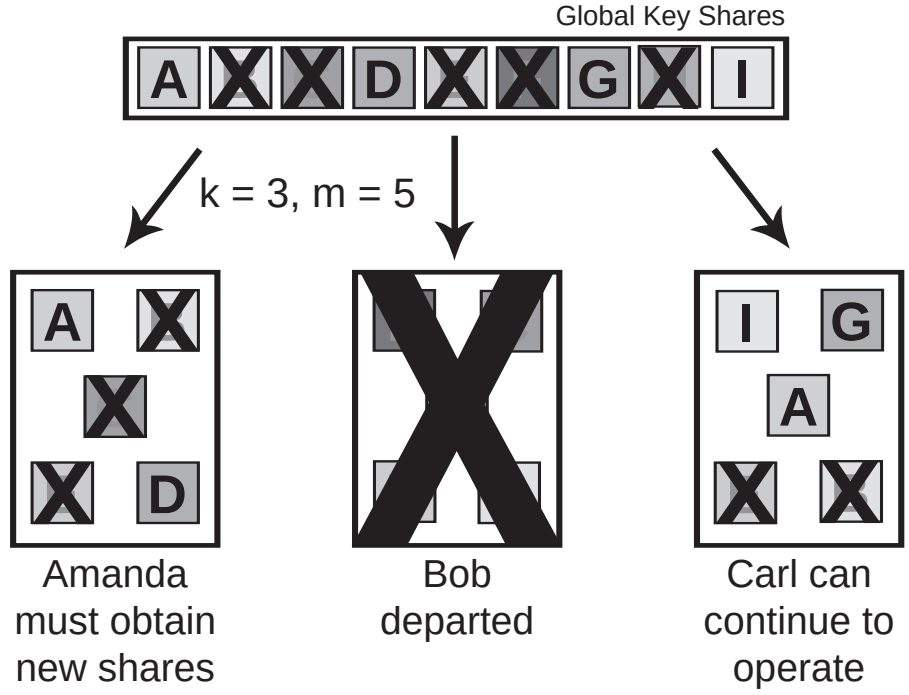


Figure 5: *Amanda Reports Lost or Stolen Handheld*

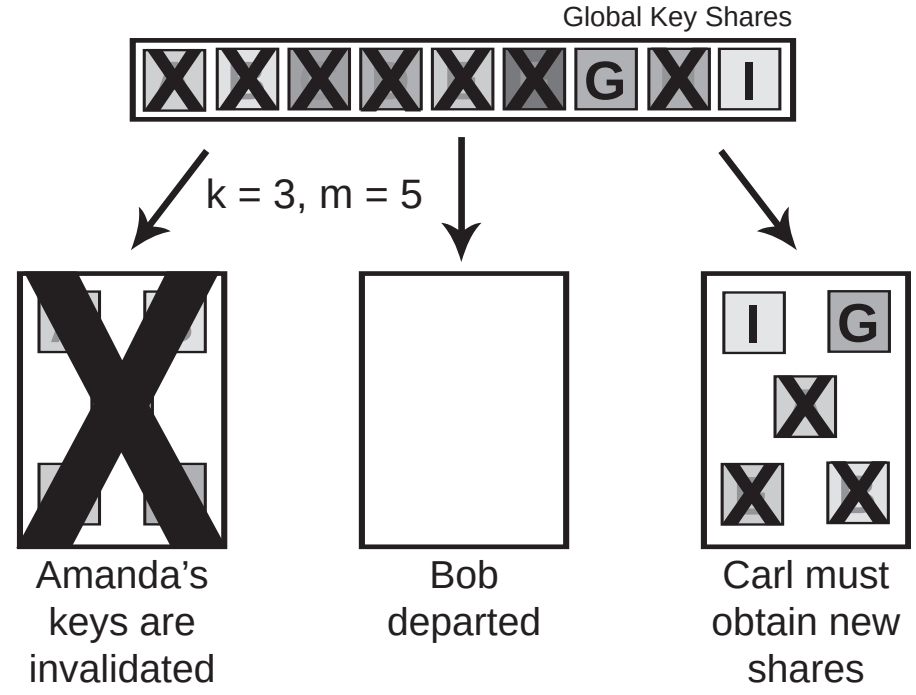


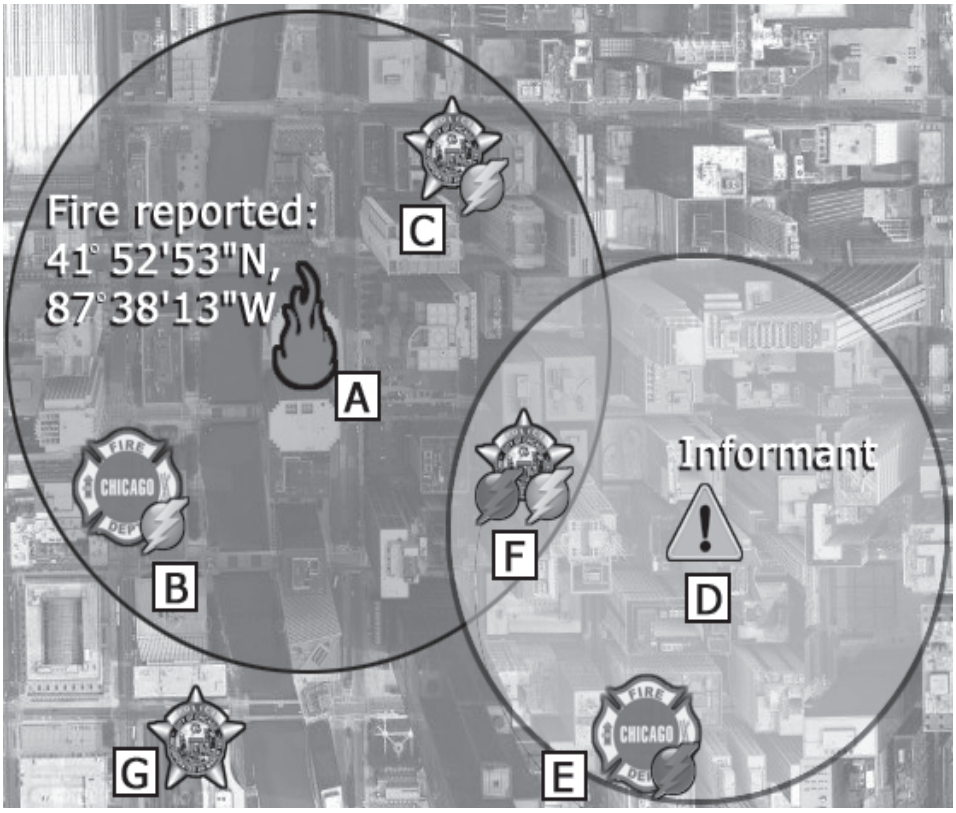
Figure 6: *Mobile Client Screenshots for the Event Reports System*

information server, which is a Postgres database with a PostGIS [5] extension that is integrated with an instance of a visualization server in an application daemon. The visualization server renders map data for visualization [6] in concert with an instance of a TIGER database [4]. When a client enters an event report region, the database triggers the insertion of a new record into a special table. Meanwhile, the event reports daemon monitors this table. If there are any new entries, the daemon creates an SMS and sends it to the target user. The heavy lifting for this mechanism is done through an extension of Postgres triggers (Figures 8 and 9 show an example for alpha-numeric and spatial range triggers), resulting in fewer queries and better performance.

Trigger support in Postgres is table-based and comparatively primitive: with *n* table triggers, an update will cause *n* operations to occur, resulting in decreased performance if updates are frequent. Also, Postgres does not provide out-of-the-box support for multi-table triggers. This becomes a problem, for example, with mixed notifications.

To address these problems, we have implemented a *trigger meta table*, which encodes relationships between trigger class identifiers and ownership, and is referenced before trigger evaluations. Consider the mixed notification: "NOTIFY me WHEN I come WITHIN 2 miles of a gas station WITH a gas price LOWER THAN $3.50." When the user's location is updated, the trigger meta table is examined on the user ID trigger class identifier. When gas prices are updated, entries in the meta table are examined on the gas station ID and the trigger class identifier. Performance is up to eight times faster than without the meta table for alpha-numeric triggers (Figure 8), and up to 10 times faster for spatial range triggers (Figure 9). Performance increases as the total number of triggers increases.

Figure 7: *An Example Event Reports Scenario*



Agent Contingency and Action Coordinator

Another application that we have built is an AC2 application, which provides a full-text, voice, and picture messaging system. Messages may be sent directly to individual clients or by *radius*. The radius message mechanism works as follows: The sender specifies his or her GPS coordinates and radius in meters within the message header. When the message is sent to the server, all agents' last known GPS coordinates are examined. The message is sent to all agents in the defined circle. Radius messaging might be useful, for example, for

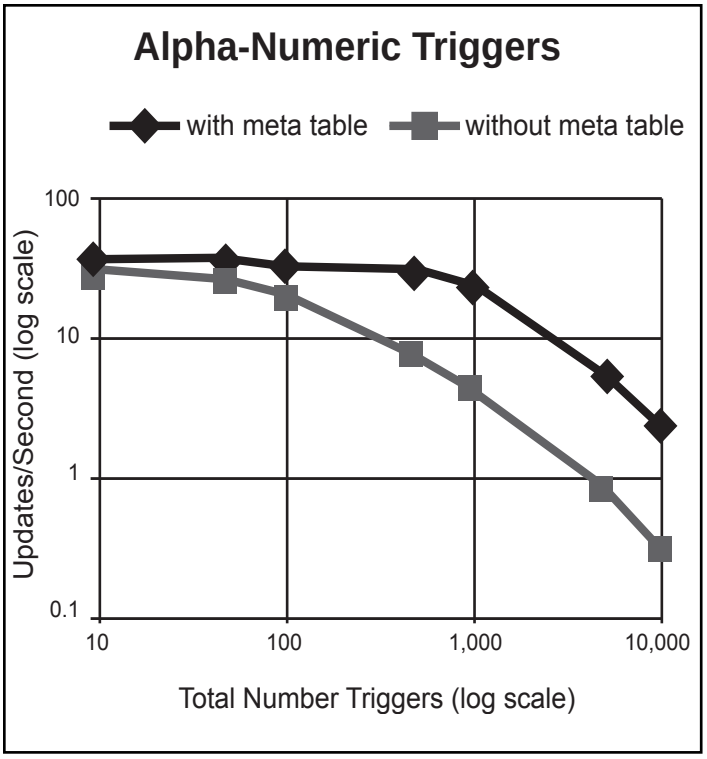


Figure 8: *Meta Table Performance Comparison for Alpha-Numeric Triggers*

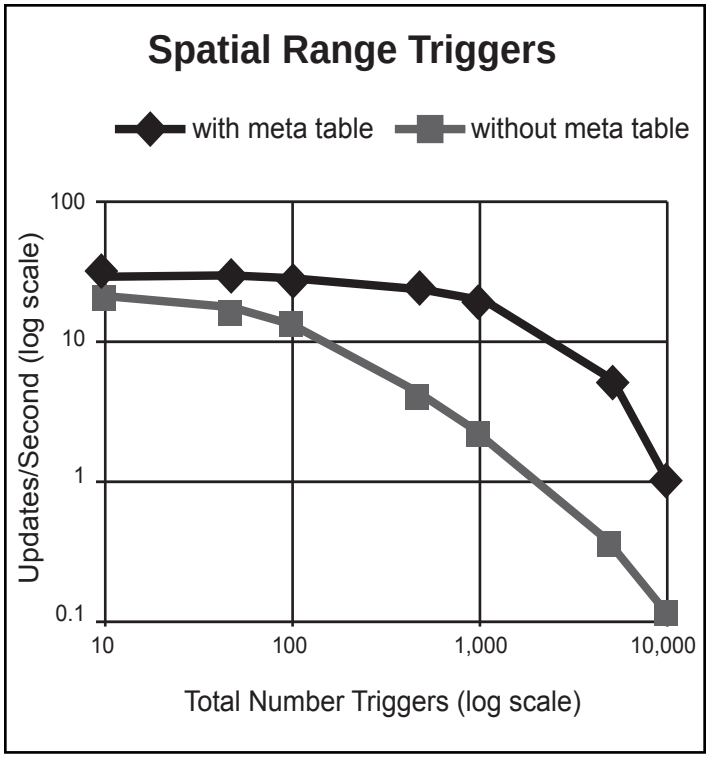


Figure 9: *Meta Table Performance Comparison for Spatial Range Triggers*

the dissemination of orders to all agents within a specific location.

Another innovative feature of AC2 is *message withdrawal*. If a client has sent a message and then later circumstances change and they no longer want the message to be read by other agents, they can withdraw the message; it will be removed from the inboxes of all agents to whom the user sent it. This is useful in situations in which agents have decided a reported incident has stopped being of interest. For example, if an agent initially reports seeing a suspicious package, but later determines that it is not a threat, they can withdraw the message to prevent confusion among the other agents. All messages—including withdrawn messages—persist in the WebBee server log so as to provide a traceable audit trail.

Conclusion

WebBee is a robust, mobile, scalable communications and coordination framework that can handle several applications at various levels of security. The challenge-response and quorum systems are scalable mobile security paradigms that are appropriate for our system. The implementation of a policy hierarchy strikes a nice balance between client situation-dependent security and future extensibility. Finally, database optimizations—like trigger meta tables and streamlined indexing—impart significant performance gains to our system.◆

References

1. Distributed System and Networks Lab at Johns Hopkins University. "SMesh." 2008 <www.smesh.org>.
2. Anderson, R. Invited Lecture. Fourth Annual Conference on Computer and Communications Security, 1997.
3. Bellare, Mihir, and Sara K. Miner. "A Forward-Secure Digital Signature Scheme." Lecture Notes in Computer Science 1666: 431-448, 1999.
4. United States Census Bureau. "Topologically Integrated Geographic Encoding and Referencing System." United States Census Bureau. 2008 <www.census.gov/geo/www/tiger>.
5. Refractions Research. "PostGIS." 2007 <http://postgis.refractions.net/>.
6. MapServer. University of Minnesota. 2008 <http://mapserver.gis.umn.edu>.

Note

1. Primary Investigators: Sugih Jamin, Zhuoqing Mao, T. V. Lakshman, Sarit Mukherjee, Jignesh Patel, and Limin Wang. Students, past and present: Brendan Blanco, Hyunseok Chang, Yun Jason Chen, Søren Dreijer, Matt England, Joe Flint, Alex Garcia, Dan Harris, Todd Hopfinger, Dan Konson, Neil Panky, Jeff Powers, Bob Sprentall, Patrick Turley, John Umbaugh, Krian Upatkoon, Wenjie Wang, Zhiheng Wang, and Byung Suk Yang.

About the Author

Sugih Jamin is an associate professor of computer science at the University of Michigan. His research interests lie in Internet measurement, protocol, as well as infrastructure design and deployment. He has earned many awards including the Sloan Foundation Fellowship, the National Science Foundation Faculty Early Career Development Award, and the White House Presidential Award. Jamin is also the chief technical officer and co-founder of Zattoo, a prominent startup that develops technology for delivering television content over the Internet. In addition to WebBee, his research at the University of Michigan includes building protocols, architectures, and mechanisms to support new uses of computer networks, as well as measuring, studying, and characterizing Internet topology and traffic.

The Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109-2121
Phone: (734) 763-1583
Fax: (734) 763-1260
E-mail: jamin@eecs.umich.edu

Constructing Change-Tolerant Systems Using Capability-Based Design

Dr. James D. Arthur and Ramya Ravichandar
Virginia Tech

Large-scale, complex emergent systems demand extended development life cycles. Unfortunately, the inescapable introduction of change over that period of time often has a detrimental impact on quality, and tends to increase associated development costs. In this article, we describe a capability-based approach to evolving change-tolerant systems; that is, systems whose entities (or capabilities) are highly cohesive, minimally coupled, and exhibit balanced levels of abstraction.

The widespread advancements in technology have encouraged the demand for large-scale problem solving. This has resulted in substantial investments of time, money, and other resources for complex engineering projects such as hybrid communication systems, state-of-the-art defense systems, and technologically advanced aeronautics systems. Unfortunately, the expenditures are belied by the failure of such systems. Plagued by evolving needs, volatile requirements, market vagaries, technology obsolescence, and other factors of change, a large number of projects are prematurely abandoned or are catastrophic failures [1, 2, 3]. The inherent complexity of these systems, compounded by their lengthy development cycles, is further exacerbated by utilizing development methods that are hostile to change. Moreover, this complexity often results in emergent behavior [4] that is unexpected. For example, the introduction of a new functionality in the system can result in unanticipated interactions with other existing components that can be detrimental to the overall system functionality.

More recently, techniques such as the performance-based specifications (PBSs) [5, 6] and capability-based acquisition (CBA) [7] are being utilized to mitigate change in large-scale systems. PBSs are requirements describing the outcome expected of a system from a high-level perspective. The less detailed nature of these specifications provides latitude for incorporating appropriate design techniques and new technologies. Similarly, CBA is expected to accommodate change and produce systems with relevant capability and current technology. It does so by both delaying requirement specifications in the software development cycle and allowing time for a promising technology to mature so that it can be integrated into the software system. However, the PBS and CBA approaches lack a scientific procedure for deriving system specifications from an initial set of user needs. Moreover, they neglect to define the level of abstraction at which a specification or a capability is to be described. Thus, these approaches propose solutions that are not definitive, comprehensive, or mature enough to accommodate change or benefit the development process for complex emergent systems.

In order to function acceptably over time, complex emergent systems must accommodate the effect of dynamic factors—such as varying stakeholder expectations, changing user needs, advancing technology, scheduling constraints, and market demands—during their lengthy development periods. We conjecture that these changes can be achieved with minimum impact if systems are architected using aggregates that are embedded with change-tolerant characteristics. Such aggregates are defined as *capabilities*.

> *"... changes can be achieved with minimum impact if systems are architected using aggregates that are embedded with change-tolerant characteristics."*

Capabilities are functional abstractions that populate the space between needs and requirements. As such, they (a) are more rigorously defined than user needs, (b) retain crucial context information inherent to the problem space, but at the same time (c) avoid solution specification commitments ascribed to requirements. Capabilities are constructed so that they exhibit high cohesion, low coupling, and balanced abstraction levels. The property of high cohesion helps localize the impact of change to within a capability. Also, the ripple effect of change is less likely to propagate beyond the affected capability because of its reduced coupling with neighboring capabilities [8]. The balanced level of abstraction assists in understanding the embedded functionality in terms of its most relevant details [9]. Additionally, we observe that the abstraction level is related to the size of a capability; the higher the abstraction level, the greater the size of a capability [10]. From a software engineering perspective, abstractions with a smaller size are more desirable for implementation.

Capabilities are generated using a capabilities engineering (CE) process. Specifically, this approach employs a unique algorithm and a set of well-defined metric computations that exploit the principles of decomposition, abstraction, and modularity to identify functional aggregates (i.e., capabilities). Such capabilities embody the desirable software engineering attributes of high cohesion, low coupling and balanced abstraction levels. Change-tolerance is achieved through the embodiment of such attributes. The integration of the CE process with existing development paradigms, and the exploitation of enhanced traceability that accompanies it, are expected to reveal more effective methods for designing, building, and maintaining software for real-world systems. This results in a capability-based system specification that is change-tolerant, permitting a just-in-time specification of requirements, and an incremental development cycle that can span long periods of time.

The CE Process

The problem of changing requirements, especially in developing large complex systems, is well established [11]. Software development processes that are ill-equipped to accommodate change are pri-

marily afflicted with requirements volatility [12]. This phenomenon is known to increase the defect density and affect project performance resulting in schedule and cost overruns [2, 13]. Traditional requirements engineering (RE) strives to manage volatility by baselining requirements. However, the dynamics of user needs and technology advancements during the extended development periods of complex emergent systems discourage fixed requirements.

Our approach, the CE process, builds change-tolerant systems on the basis of optimal sets of capabilities. Figure 1 illustrates the two major phases of the CE process. Phase I identifies sets of capabilities based on the values of cohesion, coupling, and abstraction levels. Phase II, a part of our ongoing research, further optimizes these initial sets of capabilities to accommodate schedule constraints and technology advancements. The CE process is discussed further in the following section.

The capabilities identification algorithm (also described in the following section) employs measures of cohesion, coupling, and abstraction to identify candidate sets of capabilities that necessarily and sufficiently embody the desired system functionality. Once identified, they can be further optimized to suit schedule and/or technology constraints; but because capabilities are formulated from user needs, our efforts required focus on needs analysis, a phase prior to requirements specification. At this point, we consider only the functional aspects of a system.

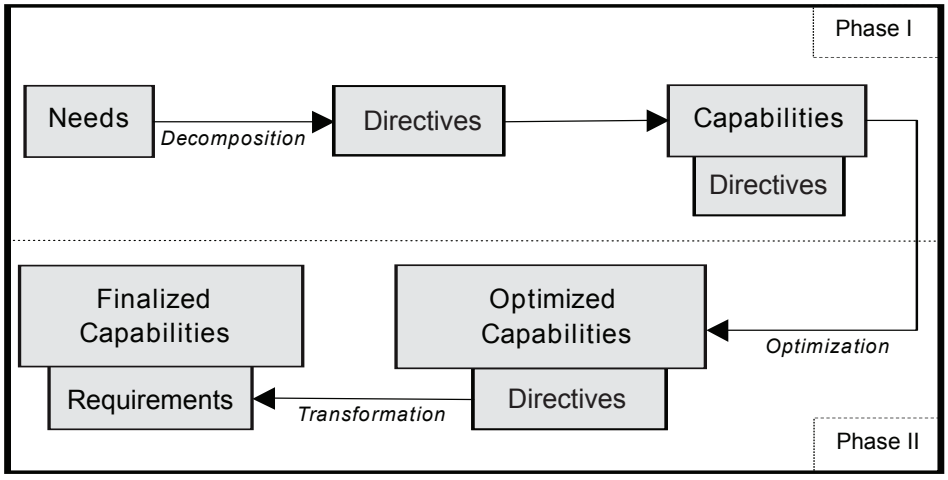


Figure 1: The *CE Process*

Computing Capabilities: The Algorithm

Capabilities are determined mathematically from a function decomposition (FD) graph (see Figure 2). This is an acyclic directed graph, implicitly derived from user needs, and represents system functionality at various levels of abstraction. The highest abstraction level, represented by the root node, connotes the mission of the system; the lowest levels of abstraction (i.e., the leaves), represent *directives*. Directives are low-level characteristics of the system formulated in the language of the problem domain. They differ from requirements in that requirements are formulated using language and terminology inherent to the more technically oriented solution domain. Thus, capabilities are identified after the elicitation of needs but prior to the formalization of technical system requirements. This unique spatial positioning permits the definition of capabilities to be independent of any particular development paradigm. We envision that by doing so, capabilities can bridge the chasm between the problem and the solution space, also described as the *complexity gap* [14]. It is recognized that this gap is responsible for information loss, misconstrued needs, and other detrimental effects that plague system development [15, 16].

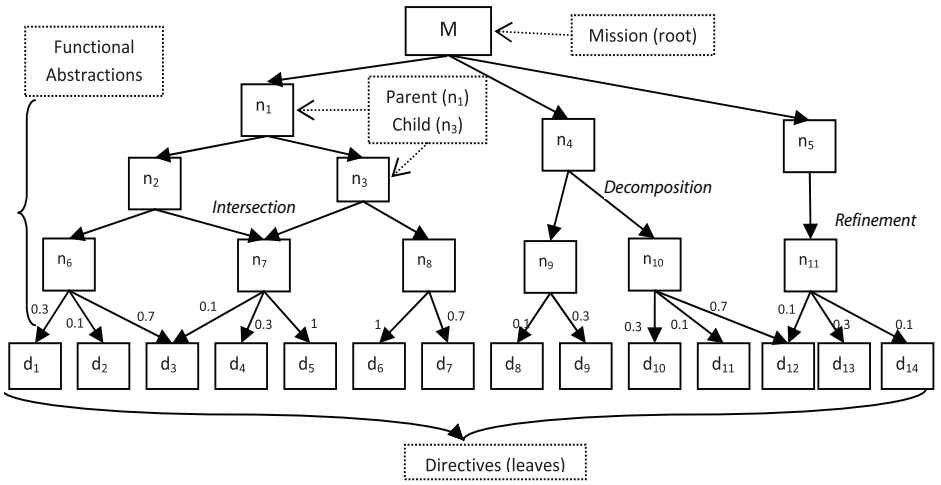
To identify capabilities, we need to examine all possible functional abstractions of a system represented in the FD graph. Intuitively, the algorithm for computing the desired set of capabilities is a five-step process that produces *slices* through the FD graph. We define a *slice* to be any subset of interior nodes of the FD graph such that their respective frontiers uniquely cover all directives. We select the slice containing the set of interior nodes that are maximally cohesive, minimally coupled, and exhibit balanced levels of abstraction. In effect, this slice contains the desired set of capabilities.

The following sub-sections outline the process for identifying the slice containing the desired set of capabilities.

Step 1: Constructing the Functional Decomposition Graph

An FD graph represents functional abstractions of the system obtained by the systematic decomposition of user needs. A need at the highest level of abstraction is the mission of the system and is represented by the root. We use the top-down philosophy to decompose the mission into functions at various levels of abstraction. We claim that a decomposition of needs is equivalent to a decomposition of functions because a need essentially represents some functionality of the system. Formally, we define an FD graph $G = (V,E)$ as an acyclic directed graph where V is the vertex set and E is the edge set. V represents the system's functionality: Leaves represent directives, the root symbolizes the mission, and internal nodes indicate system functions at various abstraction levels. Similarly, the edge set E comprises edges that depict decomposition, intersection, or refinement relationships among nodes. These edges are illustrated in Figure 2. An edge between a *parent* and its *child* nodes represents functional decomposition and implies that the

Figure 2: *Example FD Graph G = (V,E)*



Impact of Directive Omission	Description of Impact on Associated Parent Task	Relevance of Directive
Catastrophic	Task failure	1.00
Critical	Task success questionable	0.70
Marginal	Reduction in performance	0.30
Negligible	Non-operational impact	0.10

Table 1: *Relevance Values*

functionality of the child is a proper subset of the parent's functionality. Only internal (non-leaf) nodes with an outdegree of at least 2 can have valid decomposition edges with their children. The refinement edge is used when there is a need to express a node's functionality with more clarity, say, by furnishing additional details. A node with an outdegree of 1 symbolizes this type of relationship with its child node. To indicate the commonalities between functions defined at the same level of abstraction, the intersection edge is used. Hence, a child node with an indegree greater than 1 represents a functionality common to all its parent nodes. The FD graph utilizes these definitions to provide a structured top-down representation of system functionality, thereby facilitating the formulation of capabilities in terms of their cohesion, coupling, and abstraction values. We discuss those computational mechanics next.

Step 2: Computing Cohesion and Coupling Values

Only interior nodes are considered as capability candidates. For each interior node, its cohesion value is directly proportional to how important its children nodes are to achieving its defined functionality. Coupling, on the other hand, is a pair-wise relationship between two interior nodes and reflects the probability that a change in one node will have an impact on the other. The cohesion value for each node and the coupling value for each set of pair-wise nodes are computed using the FD graph *G,* and these measures are described next.

- **Cohesion.** The cohesion of a node is computed as an average of the relevance values of the participating directives. The relevance values are assigned based on the values listed in Table 1. However, we make a distinction between the parent and ancestor nodes of a directive. In order to reduce the need for user input, we elicit the relevance value of a directive only with respect to its parent node. Figure 2 illustrates relevance values of directives to their parents.

  Assuming that each directive can be associated with a unique parent node (the validity of this assumption is established in [17]), then the cohesion for any node *n* can be computed (as shown in Figure 3).
- **Coupling.** To measure coupling, we need information about dependencies between system functionalities. By virtue of its construction, the structure of the FD graph represents the relations between different aggregates. In particular, we compute coupling between two nodes in terms of their directives. Two directives are said to be coupled if a change in one affects the other. We compute this effect as the probability that such a change occurs and propagates along the shortest path between them. Note that the coupling measure is asymmetric.

  For two nodes *p* and *q,* the coupling between them is computed (as shown in Figure 4).

Figure 3: *Equation for Computing Node Cohesion*

(a) if node *n* has *only* directives as its children, then its cohesion is the arithmetic mean of the relevance values of the associated directives, i.e.:

$$Coh(n) = \frac{\sum_{\substack{\text{for each directive } i \\ \text{associated with node } n}} (\text{relevance value for directive } i)}{\text{total \# of directives associated with node } n}$$

(b) for all other nodes:

$$Coh(n) = \frac{\sum_{\substack{\text{for each immediate child } i \\ \text{associated with node } n}} [(\text{\# of directives associated with child } i) * (\text{cohesion of child } i)]}{\sum_{\substack{\text{for each immediate child } i \\ \text{associated with node } n}} (\text{\# of directives associated with child } i)}$$

Figure 4: *Equation for Computing Coupling Between Nodes*

$$Cp_{node}(p,q) = \frac{\sum_{\substack{\text{for each pairwise directive } i \text{ and } j \\ \text{associated with nodes } p \text{ and } q, \text{ respectively}}} (\text{coupling between directive i and directive j})}{\left(\begin{matrix}\text{total number of directives} \\ \text{associated with node } p\end{matrix}\right) * \left(\begin{matrix}\text{total number of directives} \\ \text{associated with node } q\end{matrix}\right)}$$

where the coupling between two directives *i* and *j* is computed as:

$$Cp_{directive}(i,j) = \frac{(\text{probability that directive j will change})}{\left(\begin{matrix}\text{length of shortest path} \\ \text{connecting directive i and directive } j\end{matrix}\right)}$$

and where the probability that directive *j* will change is computed as:

$$PrbChg(j) = \frac{1}{\left(\begin{matrix}\text{total \# of directives associated} \\ \text{with parent node of directive } j\end{matrix}\right)}$$

Step 3: Identifying the Candidate Set of Slices

Recall that slices are sets of nodes that necessarily and sufficiently cover all directives identified in the FD graph. Moreover, no slice contains the mission node (*M*). We compute all possible slices and then rank them. The first ranking (high to low) is based on the average cohesion of each slice's constituent nodes; the second ranking (low to high) is based on the average coupling of each slice's constituent nodes. The top 10 slices (an arbitrary count) common to both sets are then

chosen to form the *pruned* candidate set and represent those slices that have the highest average cohesion and lowest average coupling.

Step 4: Computing Balanced Abstraction Levels

In the next step, we individually examine each of the 10 slices with the objective of iteratively decomposing constituent nodes to achieve a balanced level of implementation abstraction. The decomposition process consists of replacing a parent node with its children nodes. We observe that as nodes are decomposed the abstraction level becomes lower—that is, the node sizes decrease but the coupling values increase (size is the number of directives associated with an interior node). We strive to identify nodes of reduced sizes in line with the principles of modularization, but only if the increase in coupling is acceptable. There are two possible scenarios when attempting to lower the abstraction level of a node: The replacement (children) nodes have lower-level common functionality, or they have no common functionality. Referring again to the FD Graph in Figure 2, suppose that one of the candidate slices is $\{n_1, n_4, n_5\}$.

- **Common Functionality.** Assume that the size of n_1 is too large, and hence, we attempt to reduce its abstraction level to its children, viz. n_2 and n_3, which are of a relatively smaller size. We observe, however, that these nodes share a common functionality represented by n_7. This implies that one of the links, (n_2, n_7) or (n_3, n_7), needs to be broken in order to implement n_7 as a part of a single-parent capability. Let (n_2, n_7) be broken, and n_7 be implemented as a part of n_3. Consequently, capabilities n_2 and n_3 are content-coupled [16] because n_2 may attempt to manipulate the n_7 part embodied in n_3. Thus, lowering the abstraction level of n_1 results in capabilities of decreased sizes, but with increased coupling.
- **No Common Functionality.** Now we consider the reduction of n_4 to smaller-sized nodes, n_9 and n_{10}. Note that the proposed reduction has no commonalities. We observe that there is a marginal increase in coupling, but that nodes n_9 and n_{10} are of smaller sizes when compared to n_4. Thus, we choose n_9 and n_{10} over their parent n_4. We are willing to accommodate this negligible increase in coupling for the convenience of increased modularity, a decision based, in part, on subjective evaluation.

Hence, we iteratively compute the appropriate abstraction level for each node in the set of slices identified in Step 3, and perform the appropriate decomposition substitutions. Because the nodes selected for abstraction balancing are in the set of slices resulting from Step 3, they also exhibit high cohesion and low coupling.

Step 5: Selecting the "Optimal" Set of Capabilities

As the final step, we re-compute the average coupling and cohesion values for each of the 10 slices. *The slice having the best balance between high cohesion and low coupling is selected as the set of capabilities for the system.*

A Validation of the Current Work

We empirically tested the hypothesis that a system design based on capabilities is more change-tolerant than a design generated from the traditional RE approach. More specifically, we examined the impact of changing needs on the RE- and CE-based designs of a course evaluation system [17]. The original high-level design of this system is based on an RE approach and is termed RE-based design. The CE-based design was constructed using a capabilities approach for the system. To determine the optimal capability set, we constructed an FD graph and then applied the algorithm described earlier. This resulted in a total of 1,495 slices, from which the slice containing the set of nodes exhibiting the highest average cohesion, lowest average coupling, and a balanced abstraction level was selected as the desired capabilities of the course evaluation system. The CE-based design was constructed based on the chosen capability set.

The RE- and CE-based designs were then subjected to various changes in needs. In particular, we examined the impact of six different needs' changes on the course evaluation system. An example of a need change is, "The users need information about the handicapped-accessible facilities for courses taught in Room X." We propagated each change on the RE- and CE-based designs and recorded the number of affected classes. We performed the Wilcoxon Signed-Rank test, the non-parametric alternative to the paired t-test, which results in a P-value of 0.018. The P-value indicates the probability that the population medians of the number of affected classes in the RE- and CE-based designs are different because of chance. The very small P-value compels us to reject the null hypothesis that the change-tolerance of the system is indifferent to either the RE or the CE approach. Thus, the alternate hypothesis that the number of impacted classes in the CE-based design is significantly less than that of the RE-based design is true. This result is in agreement with our research claim that the change-tolerance of a system improves with the use of a design based on capabilities.

Summary

The current and proposed work addresses several issues associated with the design, evolution, and emergent behavior of large-scale, real-world software systems. As stated in this article, CE provides a first-level architectural decomposition of the software system. Modularity and reasoned aggregation are cornerstones for identifying change-tolerant functional units. The underlying algorithm, employing metric-based computations, extends needs analysis to produce sets of capabilities enumerating multiple composition choices and, at the same time, indicates the advantages/disadvantages of selecting one set over the other. The use of capabilities also permits the delayed commitment of needs to requirements which, in turn, support the integration of new technology throughout the (extended) software development effort. Moreover, because capabilities are designed to be loosely coupled, they facilitate emergent behavior through the addition/deletion of functionality as new operational conditions and constraints evolve. Finally, we expect capabilities to support earlier architectural analysis, leading to the design of systems that better accommodate non-functional requirements like performance, security, and reliability.◆

References

1. Deal, D.W. "Beyond the Widget: Columbia Accident Lessons Affirmed." Air Space Power XVIII 2 (2004): 31-50.
2. Glass, R.L. Software Runaways: Monumental Software Disasters. Upper Saddle River, NJ: Prentice Hall, 1998.
3. Jazequel, J.M., and B. Meyer. "Design By Contract: The Lessons of Ariane." Computer 30 (1997): 129-130.
4. Heylighen, F. Self-Organization, Emergence, and the Architecture of Complexity. Proc. of the First European Conference on System Science. Paris, France, 1989: 23-32.
5. Tull, G.A. Guide for the Preparation and Use of Performance Specifications. Department of Defense (DoD) AMC Pamphlet. Alexandria, VA., 1999: 715-17.
6. DoD. Guidebook for Performance-Based Services Acquisition. DoD

Acquisition, Technology, and Logistics. 2002 <www.acq.osd.mil/dpap/Docs/pbsaguide010201.pdf>.

7. Montroll, M., and E. McDermott. "Capability-Based Acquisition in the Missile Defense Agency." Storming Media (2003) <www.stormingmedia.us/82/8242/A824224.html>.
8. Haney, F.M. Module Connection Analysis – A Tool for Scheduling Software Debugging Activities. Proc. of AFIPS Joint Computer Conference. Boston, MA., 1972: 173-179.
9. Parnas, D.L., P. Clements, and D. Weiss. The Modular Structure of Complex Systems. Proc. of the Seventh International Conference on Software Engineering. Orlando FL., 1984: 408-417.
10. Ravichandar, R., J.D. Arthur, and R.P. Broadwater. Reconciling Synthesis and Decomposition: A Composite Approach to Capability Identification. Proc. of the 14th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. Tucson, AZ., 2007: 287-296.
11. Curtis, B., H. Krasner, and N. Iscoe. "A Field Study of the Software Design Process for Large Systems." Communications of the ACM 31 (1998): 1268-1287.
12. Harker, S.D.P., K.D. Eason, and J.E. Dobson. The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering. Proc. of the IEEE International Symposium on Requirements Engineering. San Diego, CA., 1993: 266-272.
13. Zowghi, D., and N. Nurmuliani. A Study of the Impact of Requirements Volatility on Software Project Performance. Proc. of the Ninth Asia-Pacific Software Engineering Conference. Gold Coast, Australia, Dec. 2002: 3.
14. Racoon, L.B.S. "The Complexity Gap." ACM SIGSOFT Software Engineering Notes. 1995.
15. Lauesen, S., and O. Vinter. "Preventing Requirements Defects: An Experiment in Process Improvement." Requirements Engineering 6 (2001): 37-50.
16. Zave, P., and M. Jackson. "Four Dark Corners of Requirements Engineering." ACM Transactions on Software Engineering and Methodology 6 (1996): 1-30.
17. Ravichandar, R., J.D. Arthur, S.A. Bohner, and D.P. Tegarden. "Introducing Change-Tolerance Through Capabilities-Based Design: An Empirical Analysis." Journal of Software Maintenance and Evolution (20.2) 2008: 135-170.

About the Authors

James D. Arthur, Ph.D., is an associate professor of computer science at Virginia Tech. His research interests include software engineering and requirements engineering as well as methods and methodologies supporting software quality assessment. Arthur has served as an advisor to the U.S. Navy Commonality Working Group, as chair of the Education Panel for National Software Council Workshop, and was guest editor for the *Annals of Software Engineering*. He has master's and doctorate degrees in computer science from Purdue University, and a master's degree in mathematics from the University of North Carolina at Greensboro.

Virginia Tech
Department of Computer Science
2050 Torgersen Hall
Blacksburg, VA 24060
Phone: (540) 231-7538
E-mail: arthur@vt.edu

Ramya Ravichandar is currently a doctoral candidate at Virginia Tech. Her research interests include large-scale system analysis, requirements engineering, change-tolerant systems, software measurement, and impact analysis. Ravichandar is a member of the Institute of Electrical and Electronics Engineers Computer Society.

Virginia Tech
Department of Computer Science
2050 Torgersen Hall
Blacksburg, VA 24060
Phone: (540) 231-7542
E-mail: ramyar@vt.edu

DoD Business Mission Area Service-Oriented Architecture to Support Business Transformation

Dennis E. Wisnosky, Dimitry Feldshteyn, Wil Mancuso, Al (Edward) Gough, Eric J. Riutort, and Paul Strassman
Department of Defense

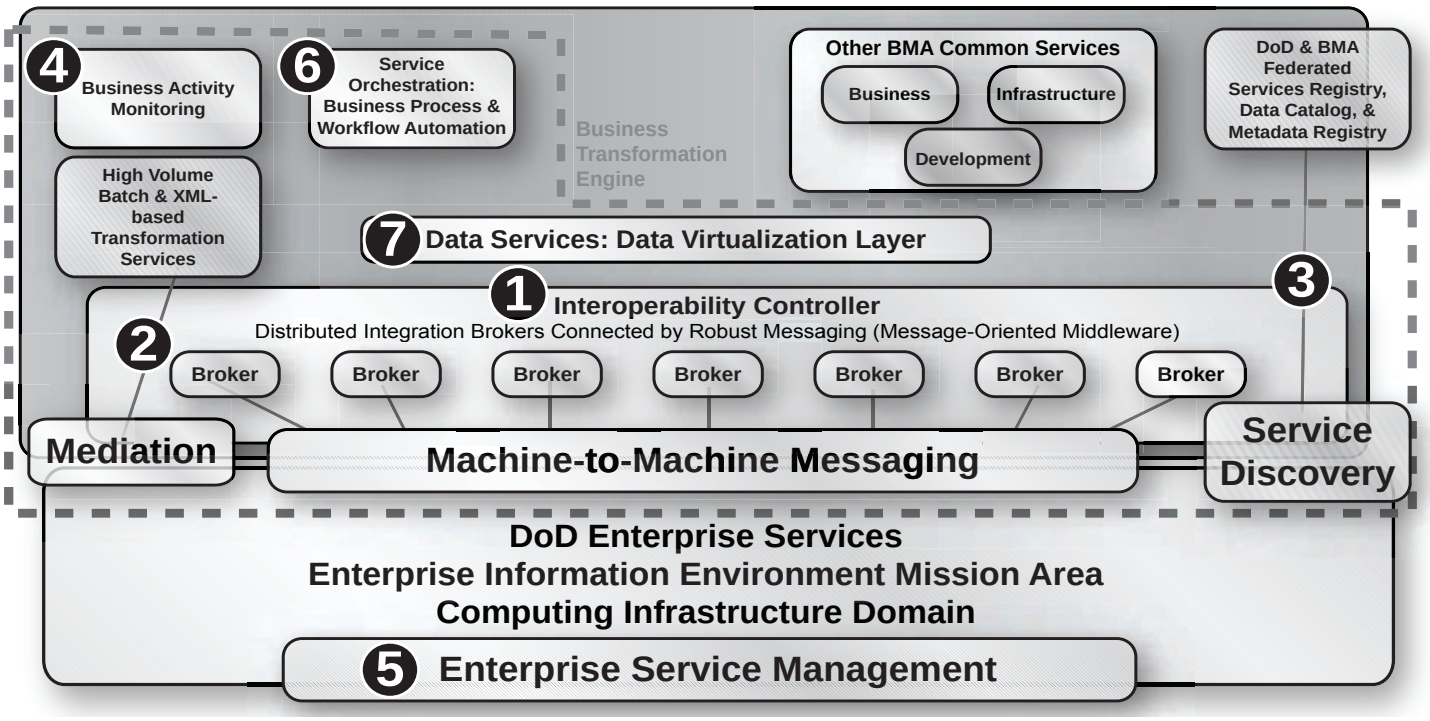
The Department of Defense (DoD) Business Mission Area (BMA) accounts for roughly half of the DoD Information Technology (IT) budget. Many of the DoD's business systems have been in use for years and are straining to support the agility of business operations necessary today. As well, many new systems are being developed on such a scale that it takes nearly a decade to produce the first results. A potential answer to this situation is delivering business capabilities through a service-oriented architecture (SOA)[1]. Much of the private sector is rapidly moving in this direction. The question is, will it work for the DoD? This article is about the results of market research conducted by the BMA Chief Technical Officer (CTO) and Chief Architect (CA) over a period of about six months to learn about state-of-the-art SOA and what the DoD can count on from SOA vendors to deliver both business services and SOA infrastructure in the near- to mid-term.

The DoD is perhaps the largest and most complex organization in the world, employing nearly 1.4 million people and holding approximately $1.4 trillion in assets. IT spending for business support activities in the DoD BMA—funds to operate, maintain, and modernize business systems—comprise $15.7 billion of annual DoD IT spending, roughly equal to the rest of the federal government.

While the DoD has long been acknowledged for its premier warfighting capabilities, fragmentation of financial and business management practices leaves the DoD vulnerable to waste, fraud, and abuse, as well as risk of failure on attempts to build larger, more complex systems. To support the DoD mission and the changing nature of the threats to which the federal government must respond, the DoD BMA is engaged in a massive business transformation. It must modernize and become agile in order to support 21st century national security requirements. The BMA CTO evaluated DoD BMA enterprise processes and associated systems—including human resource and personnel management, supply chain, and logistics, as well as financial and accounting management functions—to determine the best strategy for achieving agility. After analysis and assessment of BMA objectives and study of the overall direction for IT within the DoD, the strategy selected to move the BMA forward is the adoption of an SOA.

The U.S. Government Accountability Office describes an SOA as an "approach for sharing functions and applications across an organization by designing them as discrete, reusable, business-oriented services" [1]. Most importantly, an SOA is a mechanism by which business capabilities can be aligned with the technical infrastructure in support of an agile business strategy. The DoD's SOA vision calls for such alignment through an architecture of discrete components called *services* delivering business capabilities deployed in a supportive infrastructure designed for this purpose. The Office of the BMA CTO and CA collaborated with the chief information officers (CIOs) representing the military services, defense agencies, combatant commands, mission areas, and DoD Enterprise Services to develop an SOA strategy, including a supporting environ-

Figure 1: *The Business Transformation Infrastructure*



ment termed the Business Operating Environment (BOE). The BOE leverages industry best practices to federate technical architectures, develop capability requirements, and support the delivery of portfolios of business capabilities based on collections of atomic and/or composite service orchestrations. The BOE is defined in [2], which details the infrastructure component of the BOE and the business transformation infrastructure (BTI), shown in Figure 1 (on the previous page). Some functions of the BTI will be met through and built upon the DoD Global Information Grid Enterprise Services. The technical core of the BTI, designated the Business Transformation Engine (BTE), is to be built from commercially available products.

To assess the feasibility of this strategy, the BMA CTO conducted market research into maturity and readiness to support this strategy in SOA technologies from more than 30 organizations. These had survived a preliminary screening to ensure that they were realistic and relevant. The technical research included all components of the BTE, and was conducted in accordance with departmental regulations guiding pre-acquisition market research. The organizations provided live demonstrations of their development, test, operational, and production environments. CIO offices from each military service and many defense agencies were invited to attend and participate in the presentations. This article provides research conclusions across the BTE components (numbered in Figure 1), as well as SOA information assurance and governance.

Industry Readiness to Support Key BTI Capabilities

The research approach gathered data to correspond to key technical capabilities required to build the BTI and included an assessment of the industry's maturity in providing tools to support these technical capabilities. The research did not consider SOA technologies not relevant to the BTI. In this section, we present the assessment.

Interoperability Controller

The interoperability controller component of the BTI is a pattern or foundation architecture for brokering, routing, and processing messages and service invocations within an SOA. It consists of an extensible set of integration brokers interconnected on the network by robust messaging middleware. The research looked at products supporting this pattern, examining them for a number of characteristics, including support for indirection and interception, loose coupling, scalability, and robustness. In general, the products that most closely support this pattern are enterprise service bus (ESB) products, as well as enterprise application integration and message-oriented middleware through composition.

The market research shows that the state of industry products as reasonably mature and can support the implementation of the BMA vision for the BTI's interoperability controller component. The message-oriented middleware and enterprise application integration product vendors have been working in this direction through many generations of products. The ESB vendors have built on this experience to provide an enterprise-wide solution, though often with proprietary features. The challenge with the latter is to build a standards-based SOA that leverages the success of Web technologies rather than an ESB-based solution that provides some aspects of SOA but could lead to over-dependence on a particular vendor's technology.

"The challenge ... is to build a standards-based SOA that leverages the success of Web technologies rather than an ESB-based solution that provides some aspects of SOA but could lead to over-dependence on a particular vendor's technology."

Mediation – Standard and High Volume

While increasingly more BMA systems and data sources will communicate natively in terms of standard message sets and vocabularies, there is a short-term need for mediation of information exchanges, translating, and transforming messages between information providers and consumers. The research found good support of this pattern in both the standard and high-volume variations. Many vendors, (such as Fiorano, BEA Systems, IBM, Iona, Tibco, and webMethods) are producing capable transformation engines, especially those focused on eXtensible Markup Language (XML) messaging and the use of Extensible Stylesheet Language Transformation engines. For high volume, Ab Initio, with its advanced parallel processing capabilities, allows for the development of high-performance, straight-through mediation services. However, the vision for dynamic generation of transformations on a semantic mediation basis was found to still be a future capability. The semantic technology needed is immature, so semantic tools from companies like Revelytix and IBM are best suited for supporting development time activities, with semantics being early-bound into runtime environments.

Service Discovery and Metadata Registries

The BMA's approach to SOA calls for metadata registries and repositories supporting the discovery of services and information assets. DoD registries are built around Organization for the Advancement of Structured Information standards, such as Universal Description and Discovery Interface (UDDI) and electronic business XML (ebXML) Registry Information Model and Registry Services (including a UDDI service registry), a Metadata Registry that contains the DoD's structural and semantic metadata, and an enterprise catalog containing DoD specification metadata to support discovery of information assets. Given the DoD's size and the likely need to federate registries, the BMA included this category in the market research.

Many of the vendors in the market research provide UDDI service registries, notably Systinet, now a part of HP. Many vendors include UDDI capability (e.g., IBM, BEA, Software AG), with a number of vendors using Systinet. Many vendors also include metadata management capabilities and repository components (e.g., Fiorano, Lombardi), while others such as Revelytix specialize around semantic metadata. The DoD's metadata discovery specification is not directly supported by vendors, though those that support the ebXML architecture can act as enterprise catalog instances.

Business Activity Monitoring

Business activity monitoring (BAM)

allows management of an SOA in business terms. Market research found that BAM is still in early development. There are many vendors providing BAM functionality coming from diverse industry segments. Application integration and enterprise software vendors (BEA, Fiorano, IBM, IONA Technologies, Microsoft, Oracle, SAP, TIBCO, webMethods, etc.) are extending existing assets and acquiring additional capabilities in order to support BAM. Business intelligence vendors (Business Objects, Cognos, Software AG, etc.) are working to adapt technology and incorporate business rules engines into their solutions to support real-time BAM operations. There is also a set of pure-play BAM providers who focus on complete BAM solutions. Overall, the research found little standardization across vendor implementations, making true interoperability difficult to achieve on an enterprise level. The most common uses found in the research revolve more around project-based, application-specific uses rather than as general enterprise infrastructure.

Enterprise Services Management

Enterprise services management (ESM) provides for managing the service life cycle and is the foundation for SOA runtime governance. Market research found a limited number of SOA ESM vendors. The main vendors (e.g., IBM, Hewlett-Packard) possess strong portfolios in traditional network management and integrated service management markets that they have extended to ESM. Most of the tools researched include feature sets spanning the range from low-level IT service management to the higher-level business management needs, and the differences are more in terms of focus. Often, a more comprehensive solution can be composed by combining products (e.g., AmberPoint and HP OpenView SOA Manager).

Business Process and Workflow Automation (Business Process Modeling, Execution, and Monitoring)

The BTI must provide for the modeling and execution of business processes through the orchestration of services, and the monitoring of those business processes. While the research found that there are still many proprietary modeling offerings, there is considerable convergence around Business Process Modeling Notation (BPMN). The research also found strong support across vendors for the Business Process Execution Language standard, though there is also emerging support for direct execution of BPMN through the use of the XML Process Definition Language, an XML serialization of BPMN. Many vendors also provide the needed monitoring of those processes at runtime, often building on extensive experience with network and application monitoring capabilities. Still further in the future are tools with semantic continuity from modeling to execution in the business process arena; however, the research did find that what already exists is maturing rapidly, and can provide a base for implementing the BTI. Perhaps surprisingly, not a single vendor included the Unified Modeling Language in either its list of product offerings relative to an SOA, or as a tool that it uses in its SOA engagements.

"... the DoD is making IA services a part of the Net-Centric Core Enterprise Services so that security is ubiquitous, well-tested, and a part of the infrastructure."

Data Virtualization and Data Services

Among virtualization trends, virtualizing data sources has emerged as a real-world capability, and is a key component of the BMA SOA vision in which a virtual data store makes information from many sources available in real time without a physical store. The vendors include Composite Software, Red Hat Metamatrix, IBM, and Streambase.

The BMA research found that overall data virtualization and associated data services have matured to the point that there are many cases where they can produce high-performance and robust data sources and services to be used in a net-centric environment, significantly reducing the latency in data availability to business analysts and decision makers who do not need to wait for the periodic load of a data warehouse or data mart.

Information Assurance for SOA

An SOA introduces new information assurance (IA) challenges. The interoperability and extended, net-centric data sharing capabilities enabled by SOAs are themselves potential points of vulnerability. A compromised service registry provides an attacker with a detailed map of the operations and capabilities of an organization. Standards and standard protocols narrow the range of network capabilities that an attacker must subvert, and success wins wide access. Deploying an SOA in a responsible fashion must consider the effects of information warfare in addition to other planning. Only through such IA diligence will the DoD be able to truly realize the savings and benefits that an SOA promises for a large, geographically dispersed organization that must operate in the face of the exigencies of war. Additionally, SOAs must also meet old IA challenges including reliability, availability, and non-repudiation. An SOA does not relieve implementers of the responsibility for solid engineering in areas of platforms, networking, backups, and auditing. Past best practices and standards must be brought to bear on SOA implementations as well as traditional ones.

As would be expected, the DoD is making IA services a part of the Net-Centric Core Enterprise Services so that security is ubiquitous, well-tested, and a part of the infrastructure. An SOA provides the possibility to externalize security as a common, cross-cutting set of capabilities, themselves presented as services. In this way, each application or program does not have to master the complex technical capabilities required, but can declaratively define IA requirements and expect them to be honored and enforced by the infrastructure. At the same time, DoD-level IA policy can be enforced on SOA operations, including authorization control, redaction, and auditing. An SOA must also work with the DoD's Public Key Infrastructure to enable secure single sign-on, and to ensure preservation of appropriate non-repudiation characteristics as people and systems take action against DoD and BMA data assets.

The BTI is intended to embrace and extend the DoD SOA and IA foundations. During the research, the BMA team studied vendor capabilities with regard to IA and security in a number of areas. In particular, there was support for emerging

Web Services security standards and the inclusion of IA capabilities, both for enabling IA and for working with an enterprise's existing IA infrastructure.

- **Support for Web Services IA Standards.** Vendor support for the Web Services standards stack (WS-*) and related sets of XML and network IA standards—such as the WS-Security Assertion Markup Language, and the eXtensible Access Control Markup Language—is maturing rapidly along with the standards themselves. These standards are key to moving IA into the infrastructure, the SOA foundation, and enabling a declarative IA. Most of the deep stacks of SOA capability, such as those from IBM, BEA, Oracle, and Microsoft, have incorporated these standards throughout.
- **Enabling IA Infrastructure Capabilities.** Some organizations included in the research (such as AmberPoint) focus explicitly on providing SOA security capabilities. The market research found that there is a trend to make IA an integral part of SOA through provisioning, governance, and key infrastructure, such as with the BTI's interoperability controller. This holds out the promise that as an SOA is implemented in the BMA, it will not prove to be the *soft and chewy inside* of a *hard and crunchy* perimeter defense.
- **Integration With Existing IA Infrastructure.** DoD IA must be a consideration from the beginning of the life cycle. An SOA must be able to work and interoperate with IA standards, practices, and approaches developed during the DoD and U.S. intelligence community's long experience in producing networked IT systems to provide defense in depth. The market research found that there is a convergence in this arena, with the DoD looking to adopt industry and commercial best practices in IA for its solutions, and SOA vendors (included in the research) willing to meet and accommodate the stringent IA requirements of the DoD.

Governance

Governance—the means to assure that laws, regulations, and policies are met in IT operations and investments—is of key importance for the move to an SOA. An SOA introduces new challenges for IT and business governance due to solutions composed from numerous distributed services in an environment of heterogeneous ownership and control, and by enabling widespread sharing of information and capabilities. The BMA strategy for SOA governance addresses both buildtime and runtime needs.

Buildtime (Investment) Governance

The research assessed buildtime governance in the following areas:

- **Enterprise Architecture Satisfaction.** The research found that enterprise architecture tools are moving to explicitly model services, such as those from Mega Software or IBM (Telelogic). However, these tools have (at most) limited interoperability with tools used to design and develop services. These tools also provide little in the way of automated compliance checking or management of the transition between enterprise architecture models and service designs and implementations.
- **Duplication Avoidance.** The research found that this aspect of governance is provided largely by the ability of SOA development tools and environments to access service registries and repositories. This allows developers to determine whether an implementation for their service already exists. Additional metadata repository capabilities (providing further information) support this process.
- **Service Usage.** The market research found that the main mechanisms for assuring that existing services are used as appropriate are through development tools that integrate with an enterprise's service registries and repositories. These tools provide developers with service descriptions and specifications at design and build time. Many tool vendors, such as Lombardi and IBM, provide this capability.
- **Service Verification.** The market research found that there is good support for test and verify SOA services—against functional requirements and service level agreements (SLAs)—when combined with more traditional automated testing tools.
- **SLA Development.** The market research found support for capturing SLAs, but support for the actual initial development of the SLAs is more limited. System architects and designers need to pay close attention to how they develop SLAs and translate them into digital form for use by automated SLA management capabilities.

"While serious caution remains in the areas of IA and security ... the need for significant cultural change for successful SOA implementation cannot be overemphasized ..."

Runtime (Operations) Governance

Runtime governance should provide visibility into service operation allowing management of services, the ability to take corrective action (as needed) to ensure effectively uninterrupted business operations, and the capture of operation audit information. Provisioning, deploying new services, and taking old services out of operation without significant impact on business activities or overall operations, are key parts of overall runtime governance. Characteristics looked for in runtime governance include the following:

- **Operational Visibility.** Make the runtime state visible in both technical (network and machine usage) and business terms.
- **Service Management.** Monitor and manage the execution and operation of services in an SOA.
- **Policy Enforcement.** Enforce security and other policy-based constraints in a declarative fashion, external to SOA services, allowing systems to adapt quickly to changing policy circumstances without coding.
- **Auditing.** Track and record key events and actions within the SOA environment for later analysis.
- **Provisioning and Configuration Management.** Provision services for deployment in the SOA and track its configuration across changes as they occur.

Governance Conclusions

The market research found no complete solution available as a single package, but there is considerable governance capability available in the marketplace. For example, in the area of provisioning and configuration management, the research found that SOA management tools provide some of this capability,

but may need to be joined with more traditional configuration management and deployment tools for reasonable capability. Governance capability (as required by the BMA strategy for SOA) can be provided through commercial tools, but designers must carefully assess and acquire the components from various vendors in accordance with a strong design and plan that they must create for themselves.

Conclusion

The DoD BMA has embarked on an SOA strategy. The "BMA Architecture Federation Strategy and Roadmap" provides guidance for the DoD BMA to quickly gain business value by delivering capability to support the warfighter through an SOA, while using a phased approach for transforming legacy systems. The market research performed by the BMA Office of the CTO and CA has found that industry capabilities to implement or enable the components defined in the BMA Service-Oriented Infrastructure have matured in the marketplace. While serious caution remains in the areas of IA and security, and the need for significant cultural change for successful SOA implementation cannot be overemphasized, it is clear that it is feasible for an enterprise the size of the DoD to move forward on implementing an SOA and to realize the business benefits of agility, interoperability, and net-centric data sharing that an SOA provides.

The opinions expressed in this article are those of the authors only and in no way constitutes the policy or express direction of the DoD. For additional information about the vendors, see the online version of this article.◆

Note

1. According to the U.S. Government Accountability Office, "A service-oriented architecture is an approach for sharing functions and applications across an organization by designing them as discrete, reusable, business-oriented services. These services need to be, among other things, (1) self-contained …; (2) published and exposed as self-describing …; and (3) subscribed to via well-defined and standardized interfaces."

References

1. United States. Government Accountability Office. Defense Business Transformation: A Comprehensive Plan, Integrated Efforts, and Sustained Leadership Are Needed to Assure Success. 16 Nov. 2006 <www.gao.gov/new.items/d07229t.pdf>.
2. United States. DoD. Business Mission Area Architecture Federation Strategy and Roadmap. Ver. 2.4a. 29 Jan. 2008 <www.defenselink.mil/dbt/federation_strategy.html>.

Additional Reading

1. United States. DoD CIO. Global Information Grid Architectural Vision for a Net-Centric, Service-Oriented DoD Enterprise. 21 Mar. 2008. Ver. 1.0. June 2007 <www.defenselink.mil/cio-nii/docs/GIGArchVision.pdf>.
2. Erl, Thomas. Service-Oriented Architecture: Concepts, Technology, and Design. New Jersey: Prentice Hall PTR, 2005.

About the Authors

Dennis E. Wisnosky's responsibilities include guiding development of Federated Architectures and SOA within the DoD's BMA. He is recognized as the father of the Integrated Definition Language and the author of several books. Wisnosky holds a bachelor's degree from California University of Pennsylvania and master's degrees from both the University of Dayton and the University of Pittsburgh. He has received numerous honors for his work, including the Federal 100 Award in 2007.

DoD
Office of Business Transformation
3600 Defense Pentagon
Room 3C889A
Washington, D.C. 20301-2600
Phone: (703) 607-3440
Fax: (703) 607-2451
E-mail: dennis.wisnosky@osd.mil

Al (Edward) Gough is a senior enterprise architect supporting the DoD BMA CTO and CA. Gough is chief technology officer for CAC International Inc.'s transformation solutions group. He works on furthering the adoption and implementation of SOA across the BMA and DoD.

Dimitry Feldshteyn is a chief of SOA strategy supporting the DoD BMA CTO and CA. He is president for eMillennium Inc., and has 18 years experience working in the IT industry, 12 years of which he has spent in leadership and executive positions at Hewlett-Packard and IBM.

Eric J. Riutort is a business and IT management consultant supporting the DoD BMA CTO and CA. He works for Tech Team Government Solutions, Inc., and has 15 years of industry experience in software development, semiconductor, and automotive manufacturing. Riutort wrote the BMA's SOA governance and change management strategies.

Wil Mancuso is one of the lead SOA architects supporting the DoD BMA CTO and CA. He is the president of Information Management Solutions Consultants, Inc., and has more than 20 years of experience in industry. Mancuso developed the DoD Log EA, the Defense Logistics Agency's Integrated Data Environment, and the BMA's Common Vocabulary.

Paul A. Strassmann was DoD CIO from 1988 to 1992. In addition, he has been the chief information executive of General Foods, Kraft, Xerox, and NASA. He is presently distinguished professor of information sciences at George Mason University. He received the Distinguished Public Service Medal from the DoD as well as the Exceptional Service Medal from NASA for his work on information security and systems architecture.

Web Sites

Fault Handling and Fault Tolerance

www.eventhelix.com/RealtimeMantra/FaultHandling

EventHelix.com Inc. has several articles covering software and hardware fault handling and fault tolerance techniques. Along with articles on the basics of hardware and software as well as a description of the fault handling life cycle, you'll find techniques for making systems more tolerant to software bugs, reboot and recovery, hardware redundancy, measuring and improving computer system reliability, and calculating system availability.

Fraunhofer Center – Maryland (FC-MD)

http://fc-md.umd.edu/fcmd

The FC-MD is a non-profit applied research and technology transfer organization with the mission to advance the state-of-the-practice in software development and acquisition organizations by applying state-of-the-art research results. Along with details from their projects, publications, and training courses, you can learn about Experience Factory, their unique model for better understanding and managing your software business.

DO-178 Industry Group Homepage

www.do178site.com

DO-178B is considered the world's strictest software standard and influences other domains including medical devices, transportation, and telecommunications. The DO-178B Industry Group is the world's largest collection of avionics companies and DO-178B avionics product and services providers who share a common mission of achieving DO-178B success.

The Common Criteria (CC) Portal

www.commoncriteriaportal.org

Learn more about the CC at this internationally recognized Web portal, the driving force for the widest available mutual recognition of secure IT products. This site provides information on the status of the CC Recognition Agreement, CC and certification schemes, licensed laboratories, certified products, as well as related information, news, and events.

Vision for a Net-Centric, Service-Oriented DoD Enterprise

www.defenselink.mil/cio-nii/docs/GIGArchVision.pdf

The Department of Defense (DoD) is transforming to become a service-oriented, net-centric force. Last year, the DoD's Global Information Grid (GIG) put out a detailed *architectural vision* with the goal of promoting a unity of effort in reaching their *target state* for the development of GIG capabilities that will support future DoD missions, operations, and functions. They also provide a short, high-level, understandable description of the DoD's objective enterprise architecture.

Whose FAULT Is It, Anyway?

Way back in the 1980s, I was introduced to Moore's Law[1] in an engineering class. It says that computing power doubles approximately every 18 months to two years. This same law appears to apply to many computer-related items such as processing speed, memory capacity, and even digital camera resolution. There's a lot of similar laws for disk size, power consumption, network capacity, etc. Moore's Law has held true since its origination in 1965, and will probably hold true for at least the next decade. Every few years engineers will say that we've reached the limit of Moore's Law, but new technology keeps proving it true. The bottom line is that technology grows at an exponential rate.

My first computer, a Commodore SuperPet that I bought back in 1982, had a whopping 128 kilobytes (KB) of memory. I don't recall the clock speed, but it was a relatively slow 6502 processor that I believe was at about 1 megahertz. (As an historical point, the Commodore SuperPet *also* had a 6809 processor, and you could run dual operating systems and interpreters for Pascal, APL, FORTRAN, COBOL, plus the obligatory BASIC interpreters).

Twenty-six years later, my current laptop has three gigabytes (GBs) of memory. This pretty much follows Moore's Law, as does the processing speed of my current machine, a two gigahertz (dual processor)[2].

It appears that Moore's Law also applies to the size of the operating system. Remember MS-DOS 2.11? Back in 1983, it loaded in 64KB—and left you room to run your programs! Windows 95 (12 years later) took 50 megabytes (MB) of disk space. And now with Windows Vista, Microsoft says you need 15GB of free disk space and 512MB of memory (still following Moore's Law).

What's the point, you ask? I'm *not* Microsoft bashing. I *like* my operating system. I find it useful to run multiple applications, tons of sidebar gadgets, high-resolution graphics, and have music playing in the background.

However, with the increased program size, did you know that the chance for failure goes up, too? I know that Vista is a pretty solid operating system. I haven't had a single blue screen of death, and only about three needed hard shutdown and reboot occasions since I bought it last year (as opposed to about three-a-day from my first experiences with Windows 95, if I recall). I'm talking about the chance of failure that comes from large-scale reliance on the compliance of others. And reliance on the rapidly expanding technology raises the potential for problems.

In our office, we have a one terabyte (TB) four-disk RAID (redundant array of inexpensive disks, as named by the inventor, or occasionally known as redundant array of *independent* disks)[3] cluster. It has the very reasonable name of "Terrabyte[4]." If any two of the four-disk cluster fails, we still have a complete set of data. Until an "unexpected" failure took it totally down. Our domain name system server died—and with it down, Terrabyte was unable to grab an address. It took us a bit of time to locate the problem, and figure out how to reconfigure it. Of course, eventually, we realized that we could just plug directly into a computer. Then we realized that the permissions on the file access were based on domain authentication; so even though I could plug the device directly into my computer, it couldn't authenticate the access. Sure, it was fixable, but the delay cost several of us a bit of work. And, I admit, there was a bit of momentary panic when somebody asked, "Just in case, we *do* have a backup of it, don't we?"

We all have become dependent upon the increasing complexity of new technology. And when the technology fails, we all feel powerless. It's not like any of us can keep four or five different backups around on floppy anymore—backing up a TB RAID cluster takes some serious storage!

The point is that increased power, increased memory, and increased disk storage bring increased PPoF (Potential Points of Failure)[5]. And you need to plan for these failures.

Are you developing large-scale applications? Have you considered what to do in case the network fails? The database fails? How many backups do you have? Where are the backups located—having them in the same location really won't help in case of fire or flood, will it? Whatever technology you implement, eventually one of your users will run into a case where something goes bad, and they are going to expect you to have thought of the potential problem, and developed a contingency plan for it!

Technology lures you in—like when you're stuck in the airport, flight cancelled, you need to re-book, and you realize your cell phone is out of juice. Backup? Tried to find a pay phone lately? Kind of makes you long for the days when a spare deck of cards in your desk took care of your backup needs.

Speaking of faults, this column was almost late because the e-mail from the CROSSTALK editors reminding me my article was due was somehow misdirected into my junk mail folder. I hesitate to state how great my life would be if the other 99 percent of my daily e-mail was similarly (but faultily) misdirected. If only Outlook had an "I.Q. filter," similar to caller ID. Then, when folks complained that I never responded to their e-mail, I could say "Honest, it's not *my* fault!"

—David A. Cook, Ph.D.
The AEgis Technologies Group, Inc.
dcook@aegistg.com

Notes

1. Wikipedia. <http://en.wikipedia.org/wiki/Moore%27s_ law>. And before anybody corrects me, yes, I know that Moore's Law originally referred to the number of transistors on a chip.
2. See <http://nano-taiwan.sinica.edu.tw/2008_WinterSchool /index/Moore%27slaw%20graph2.gif> for an image of the growth in Intel Processors.
3. Wikipedia again.
4. Why the extra "r"? Because my granddaughter is named Terra, I love her, and my office was foolish enough to let me name our RAID cluster.
5. Yes, I made this one up!

Can You BACKTALK?

Here is your chance to make your point without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. These articles should provide a concise, clever, humorous, and insightful perspective on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery, and should not exceed 750 words.

For more information on how to submit your BACKTALK article, go to <www.stsc.hill.af.mil>.

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737

CrossTalk is co-sponsored by the following organizations: